USING MS-DOS KERMIT

TERMINAL EMULATION AND FILE TRANSFER

Connecting your PC to the Electronic World

SECOND EDITION

DRAFT of 25 July 1991

Christine M. Gianone

This book is lovingly dedicated to my parents, Sal and Phyllis,
who nurtured me with encouragement,
support and love throughout my life.

Contents

.

.

.

.

.

Foreword

It is with tangible joy that I introduce Christine Gianone's book, *Using MS-DOS Kermit*. This is the first book dedicated to a particular Kermit communications program, and it truly brings Kermit into the realm of the serious. Until now, users of Kermit programs have had only a thick sheaf of computer output to guide them through the intricacies of installation, communication setup, terminal emulation, file transfer, and script programming. Now MS-DOS Kermit, the most popular of all Kermit programs, has the book it deserves.

Because of its unglamorous user interface, MS-DOS Kermit may appear to the uninitiated as a no-frills product. Believe me, the frills are there, but beneath the surface where you really need them. They are found in its precision-engineered character and graphics terminal emulation, its support for every model of PC and PS/2 as well as for many non-IBM-compatible PCs, its high-speed and efficient operation, its powerful macros and scripts, and in one of the most advanced and solid implementations of the Kermit file transfer protocol to be found anywhere. Compare these aspects of MS-DOS Kermit with any commercial PC communications software package and you'll be pleasantly surprised, especially when you consider the price!

Credit for MS-DOS Kermit goes primarily to Professor Joe R. Doupnik of Utah State University, the principal author of the program. Since 1986, when he took over responsibility for MS-DOS Kermit, Joe, like all Kermit volunteer programmers, has generously donated his labors to the public through Columbia's Kermit Development and Distribution Office, which is directed by Christine, who in turn releases the Kermit software to the people of the world. Joe's work is of the highest quality, with a sensitivity to users' needs that is seldom seen among nonprofit software developers.

Chris has been responsible for the worldwide Kermit development and distribution effort for more than five years. In that time she has transformed a public-spirited but chaotic enterprise into an efficient nonprofit ‘‘business’’ as responsive to its customers as any commercial software house (more than most). Chris’s dedication, her persuasive charm and wit, her unerring good sense, and her uncanny feeling for what users *need* have kept unruly Kermit developers (like me) firmly on track, and Kermit’s countless users satisfied and loyal. And through magazine articles and press releases, seminars and lectures in far-flung areas of the world, Chris has effectively promoted the Kermit protocol into a worldwide de facto standard for file transfer.

Chris’s book is for the average PC user—a person who doesn’t care how Kermit works, how clever it is, or what goes on behind the scenes; a person whose life does not revolve around data communications, but who simply needs results. After all, Kermit is just a tool that you use to accomplish your actual goals. With this book, learning how to use the tool is a true delight.

Chris’s presentation is based on years of helping people install and use Kermit, her experience designing Kermit courses and teaching beginners, experts, and everyone in between, and on her past authorship of many of the major Kermit manuals. She is expert in the material, and has an amazing talent for paring difficult concepts down to their bare essentials and conveying them simply and directly—especially to people who don’t have that peculiar technical orientation that most PC communication packages seem to require.

If you’re an experienced PC user, you can skip right ahead to Chapter 6 to learn about Kermit itself. But if you are new to PCs and data communications, computer-shy, bewildered, or lost, read the early chapters in which Chris takes you through the mechanics of program installation and checkout, modems and cables, and the basics of MS-DOS—all with a minimum of pain and effort. After these preliminaries, *Using MS-DOS Kermit* will lead you gently and safely through the unpredictable and often hostile world of data communications without drowning you in jargon or confusing you with unnecessary detail.

Chris’s instructions are clear, concise, and step-by-step, and they are presented with a humor and sympathy rarely encountered in computer books. Useful examples appear on nearly every page, so you can build your confidence and increase your skill as you progress through the chapters. Once the material is mastered, the book remains a thoughtfully organized and valuable reference. The command summary is thorough and replete with examples. The meticulously compiled tables alone are worth the price of the book, and the Index gives you quick access to any desired topic.

Chris’s book has a unique international perspective. Her travels as ‘‘proprietor’’ of the non-proprietary Kermit protocol have taken her not only throughout the United States, but also to Europe, Japan, and most recently the Soviet Union. In response to the needs of the

non-English-speaking world, Chris has designed a groundbreaking extension to the Kermit protocol that allows for entry, display, and transfer of text in many languages; a complex topic that she describes with remarkable clarity in Chapter 13.

Another unique aspect of this book, and of MS-DOS Kermit itself, is its concern for the disabled computer user. MS-DOS Kermit, unlike most other "mass-market" PC applications, includes features that allow it to be used effectively by people who are partially sighted, blind, deaf, or physically challenged. These are presented by Chris simply and without fanfare in Chapter 15.

It has been a constant pleasure to work with two such accomplished and inspiring people as Christine and Joe. Without Joe's efforts, MS-DOS Kermit would be a dusty old has-been, and without Chris, the Kermit protocol itself would have faded into obscurity years ago. On behalf of the millions of Kermit users all over the world, my deepest thanks to them both.

Frank da Cruz
New York City, 1990

Preface

If you picked up this book and began reading it because you wondered why a muppet character's name was in the computer section of the bookstore, you'll soon discover that this book is not about muppets: *Kermit* is the name of a communications software program. In fact, it's the name of a very large family of programs that make it possible for just about any two computers anywhere to communicate with each other at the lowest possible cost. If this interests you, read on.

Using MS-DOS Kermit describes MS-DOS Kermit version 3.11 for the IBM PC, PS/2, and compatibles. With MS-DOS Kermit, a cable, and possibly a modem, you can introduce your isolated PC to the rest of world: to information and electronic mail services like CompuServe, Dow Jones News/Retrieval, or MCI Mail; to databases like BRS or DIALOG; to public data networks like SprintNet and TYMNET; and to the worldwide networks described in John Quarterman's book, *The Matrix*.[1] But most of all, you can introduce your PC to other computers: the PC on your desk to the corporate mainframe, your PC at home to the computers at work, your traveling laptop back to company headquarters, the PC in your dormitory room to the university computer center, the PC in your lab to a number-crunching supercomputer. Kermit can be your passport to the world of electronic information.

---

[1]Quarterman, John S., *The Matrix*, Digital Press (1990).

MS-DOS Kermit is only one of hundreds of Kermit programs written by public-spirited volunteers for virtually every kind of computer in existence. Each Kermit program follows the Kermit file transfer protocol, a set of rules for reliably transferring information between two computers, and many of these programs, including MS-DOS Kermit, also let you interact directly with other computers or services using your keyboard and screen for terminal emulation.

This book includes a 5.25-inch MS-DOS Kermit 3.11 diskette. If you need a 3.5-inch diskette, you may order it for a nominal fee, using the card at the back of this book. For further information about Kermit programs and how to get them, or to be placed on the mailing list for the free newsletter, *Kermit News*, write to:

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

Kermit programs are in a category by themselves. Unlike commercial software packages or "shareware," Kermit programs can be freely reproduced and shared as long as this is not done for profit. No licensing or registration is required. However, Kermit programs are not in the public domain. In general, each Kermit program includes a copyright notice to protect its special status. A large organization can potentially save itself millions of dollars by standardizing on Kermit rather than on commercial software packages, and many organizations have done just that. And, of course, private individuals save money too.

The Kermit protocol was developed in 1981 at Columbia University by Bill Catchings and Frank da Cruz at Columbia's Center for Computing Activities to fill the need for inexpensive, reliable file transfer among Columbia's diverse PCs, minicomputers, and mainframes. A decade later, Frank is still very much involved in the "Kermit project" and is the principal author of another popular Kermit program, C-Kermit.

Kermit software is contributed by programmers in many countries and has become the international de facto standard for file transfer. Today there are Kermit programs for about 400 different computers and operating systems, and the number is constantly growing. With very few exceptions, Kermit will let you transfer files between your PC and virtually any other kind of computer.

The Kermit program for the IBM PC was created by the systems programming group at Columbia University in 1982, managed by Frank. The groundwork was laid by Bill Catchings, based on his original CP/M Kermit program. Daphne Tzoar wrote the first

production version and, together with Jeff Damens, produced subsequent releases through 2.28 in 1985. In 1986, MS-DOS Kermit development was taken over by Professor Joe R. Doupnik of the Center for Atmospheric and Space Sciences and the Department of Electrical Engineering at Utah State University in Logan, Utah.

The current MS-DOS Kermit program, Version 3.11, is one of the most advanced examples of the Kermit software. It includes error-free and efficient file transfer, advanced terminal emulation, graphics, a script programming language, operation over local area networks, support for text in many languages, special features for people with physical challenges, and much more. Over the years, thanks to Joe's tireless efforts and consummate skill, and to other contributors from around the world, and to the many prerelease testers, MS-DOS Kermit has become one of the most powerful, flexible, and easy-to-use of all PC communication packages.

Using MS-DOS Kermit applies to MS-DOS Kermit Version 3.11. Many of the features described here will not be found in earlier releases. If you are running an older release, you should be able to upgrade quite easily. Use the diskette in this book, get a copy from a friend, download it from a computer bulletin board or network, or order a diskette from Columbia University.

This book is designed to teach you step by step how to make effective use of MS-DOS Kermit on the IBM PC, PS/2, and compatible personal computers. Much of what you read here will also apply to the non-IBM-compatible MS-DOS PCs that have Kermit programs available. For an overview of data communications and a complete reference on the Kermit protocol itself, see the book *Kermit, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987.

Welcome to the world of Kermit.

Acknowledgments

Special thanks to Frank da Cruz of Columbia University who opened the door for me to the world of Kermit. It is an honor to work with such a distinguished authority in the computer field. I am grateful for his recognition of my talents and the opportunities he has given me. Frank's encouragement, expertise, experience, and friendship were invaluable in the production of this book.

Special thanks also to Professor Joe R. Doupnik of Utah State University, whose skill and dedication resulted in a sophisticated software program and the subject of this book. Joe, with help from many others, has made MS-DOS Kermit a prime competitor in the PC communications software market.

Thanks to the the original MS-DOS Kermit programming staff, Daphne Tzoar, Jeff Damens, and Bill Catchings, formerly of Columbia University. And to the many others who have contributed to the program through the years: Bob Babcock (Harvard/Smithsonian Center for Astrophysics), Ron Blanford (Seattle, WA), Jack Bryans (California State University at Long Beach), Fritz Bütikofer (University of Bern, Switzerland), Edgar Butt (University of Maryland), Dick Carlton (The Open University, UK), Baruch Cochavy (IIT, Technion, Israel), Glenn Everhart (RCA, Cherry Hill, NJ), Hirofumi Fujii (Japan National Laboratory for High-Energy Physics, Tokyo), Ian Gibbons (University of Hawaii), Robert Goeke (MIT Center for Space Research), James Harvey (Indiana/Purdue University), Brian Holley (University of Cambridge, England), Terry Kennedy (Saint Peters College, NJ), Dave King (Carnegie-Mellon University), David Knoell (Basic American Food Company), Kimmo Laaksonen (Helsinki University of Technology, Finland), Mikko Laanti (University of Oulu, Finland), Christopher Lent (Cooper Union), Henrik Levkowetz (Philips Kista AG, Stockholm) Ted Medin (NOSC), Geoff Mulligan (USAF Academy), Jim Noble (Planning Research Corporation), Dan Norstedt (Stacken Computer Club, Sweden), John Nyenhuis (Purdue University), Yutaka Ogawa (Nippon Telephone and Telegraph Research Lab, Tokyo), Brian Peterson (Brigham Young University), Renne Rehmann (Switzerland), Fred Richter (Intel Corporation, Hauppauge, NY), Joseph Rock (USAF Academy), Akihiro Shirahasi (Japan National Laboratory for High-Energy Physics, Tokyo), Gregg Small (University of California at Berkeley), Dan Smith and Joe Smith (Colorado School of Mines), Andreas Stumpf (University of Stuttgart, Germany), James Sturdevant (CAP Gemini America), Glenn Trewitt (Stanford University), John Voigt (Tulane University), Joe White (Relational Technology, Inc.), Joellen Windsor (University of Arizona), and Mark Zinzow (University of Illinois). Thanks to Bill Hall of Novell Inc. for contributing the Heath character graphics files that are included on the distribution diskette. And thanks to Dr. Norman Weatherby of Columbia's Center for Population and Family Health for his early efforts in popularizing Kermit and putting it to especially good uses in far-flung corners of the world.

Grateful acknowledgments to the specialists who have contributed to the proposal for extending the Kermit protocol to accommodate international character sets—a topic that has developed into a chapter in this book: John Chandler (Harvard/Smithsonian Center for Astrophysics, USA), Alan Curtis (University of London, UK), Frank da Cruz (Columbia University, USA), Joe Doupnik (Utah State University, USA), Hirofumi Fujii (Japan National Laboratory for High-Energy Physics, Tokyo), John Klensin (Massachusetts Institute of Technology, USA), Ken-ichiro Murakami (Nippon Telephone and Telegraph Research Labs, Tokyo), Vladimir Novikov (VNIIPAS, Moscow, USSR), Jacob Palme (Stockholm University, Sweden), André Pirard (University of Liege, Belgium), Paul Placeway (Ohio State University, USA), Gisbert W. Selke (Wissenschaftliches Institut der Ortskrankenkassen, Bonn, Germany), Fridrik Skulason (University of Iceland, Reykjavik), Johan van Wingen (Leiden, Netherlands), Konstantin Vinogradov (ICSTI, Moscow, USSR), and Amanda Walker (InterCon Systems Corp., USA). And to the programmers

who are adding this protocol extension to new Kermit releases: John Chandler, Frank da Cruz, Joe Doupnik, Paul Placeway, and the "Kermit gang" at ICSTI. Thanks to Apple Computer's Higher Education Academic Development Donations program for approving a grant to help fund this effort and to Columbia University's Apple account executive, Ron Bajakian, for his assistance.

Thanks also to Dr. Robert Arnzen, Jude T. DaShiell, Robert Jaquiss, Bob Puyear, Kenneth Reid, and others for providing information and suggestions about the use of Kermit by people with physical challenges. Their comments have helped to make MS-DOS Kermit compatible with PCs and devices used by people with visual, hearing, or physical impairments.

Thanks to the people who have invited me as Kermit spokeswoman to their conferences and to their countries. These visits broadened my understanding of their data communication configurations as well as their cultures: Kohichi Nishimoto (Japan DECUS) and Ken-ichiro Murakami and Ikuo Takeuchi (Nippon Telephone and Telegraph Research Labs, Tokyo, Japan); David Gürlet (Switzerland DECUS); Juri Gornostaev, A. Butrimenko, Konstantin Vinogradov, Andrej Yuzhakov, Marina Tumanova, and Mikhail Morozov (ICSTI, Moscow, USSR); Chris Stephenson (Software Publishers Association); Bob McQueen (DECUS, Nashville, Tennessee) and guest speakers Brian Nelson (University of Toledo, Ohio), Joe Doupnik, and Frank da Cruz.

And to those not yet mentioned who were equally hospitable while I was a guest in their countries: Jean Dutertre and Jean-François Vibert (Club Kermit, Paris, France); Gisbert W. Selke (Bonn, Germany); Marina and Vladimir Morozov (Moscow, USSR); and Slava and Serge (Leningrad, USSR). Thanks also to my friend and travel agent Julie Balzer (Liberty Travel, New York) for handling the arrangements for these trips.

Thanks to the current staff of Kermit production for their dedication to the distribution of Kermit to users around the world: Maxwell Evarts, Andy Newcomb, and Ken Aparri. And to the generations of "Kermites" who came before: Robert Tschudi, Peter Howard, Lucy Lee, Kenneth Suh, Moshe Mizrahi, Takahiro Sajima, Tara Garrett, David Suarez, Sarah Lewis, Adibah Abduljalil, Rajan Vohra, Paul Pincus, David Suh, David Ho, Phil Hsu, Chun-Lun Li, Guillermo Aries, Salman Mustafa, and Kyriacos Panayioton.

And thanks to the people who have established and maintain other Kermit distribution centers throughout the world: Marc de Lyon and Frans-Jozef Springers (EARN Info Service, Netherlands), Jean Dutertre (Club Kermit, Paris, France), Juri Gornostaev and A. Butrimenko (ICSTI, Moscow, USSR), David Gürlet (DECUS, Bern, Switzerland), Ken-ichiro Murakami (Nippon Telephone and Telegraph Research Labs, Tokyo), Alan Phillips and Steve Jenkins (Lancaster University, UK), and Masamichi Ute (Science University of Tokyo, Japan).

Acknowledgments to Linda Ferreira, Gary Hanks, and Nina Frantzen of Columbia University's Division of Special Programs for inviting me to teach courses about Kermit and for preparing the class schedules and materials. The best way to thoroughly learn about a subject is to teach it to others.

Appreciation to my colleagues who have helped ease my professional transition from education to computing. Those who introduced me to the Center for Computing Activities at Columbia University: Bruce Gilchrist, Neil Sachnoff, Patricia Peters, Patrick Thompson, Sharon Moore, Terry Thompson, Anthony Bonfiglio, Fran Ovios, Bob Juckiewicz, Gary Platizky, Anna Harris, Peter Bujara, Sheila Greenbush, Robert Story, Hank Butler, Jim Cassidy, Ruth Rubinstein, Bob Bingham, Anne Simonsen, and Bill Koerber. Those who offered their consulting skills: Frank da Cruz, Maurice Matiz, Bruce Tetelman, David Millman, Don Lanini, Travis Winfrey, Ken Rossman, Vaçe Kundakçi, Jeff Damens, Mark Kennedy, Bill Chen, George Giraldi, Joel Rosenblatt, Howie Kaye, Fuat Baran, Melissa Metz, Robert Cartolano, Eric Weaver, Mark Lerner, Janet Asteroff, Jessica Gordon, Bob Miller, David Spital, and Janet Nelson.

Those who made my management training a rewarding and interesting experience: Leslie Wilkins, who has remained a dear friend, Reina Joa, Judith Weisenfeld, Carlos Jove, Becky Kaufman, Michelle Tzoar, and Sarah Bacon. And those who gave me my first introduction to data communications, even though I didn't understand the implications at the time: Bill Wojcik, Tom Hillery, Gary Williams, Peter Green, Lenny Wright, Bob Galanos, and Lon Devitt.

Thanks to Digital Press and those individuals who were part of the publication of this manuscript. Everyone involved worked skillfully under an extremely tight schedule: Mike Meehan, John Osborn, Will Buddenhagen, Chase Duffy, Beth French, Peg Tillery; the production editor, Ann Knight, and Laura Fillmore of Editorial Inc.; the thorough copy editor, Robert Fiske, and proofreaders, Barbara Jatkola and Stephani Colby; the talented book designer, Sandra Calef; the illustrator, Carol Keller; and the compositor, Frank da Cruz, a Scribe authority at Columbia's Computer Center.

Thanks to the people who reviewed this manuscript for their speed, enthusiasm, and thoughtful comments: Joe Doupnik and Frank da Cruz, two highly regarded Kermit experts, whose insights and cooperation were critical to achieving a balance between the software program and the book; Louis Santelli, for conveying the perspective of a naive user of data communications software and for his love and devotion; Sari Wilner, my dearest friend, who has read many computer software manuals and has reviewed my work before, tracing back to undergraduate college assignments; and Salvatore Gianone, my brother, who has inspired me since childhood. Although Sal is an experienced computer user, he was forced to learn far more about hardware than expected when unknowingly struggling with two pieces of inoperable equipment: a bad serial port and a broken

modem. Unfortunately, this scenario is not as uncommon as you might think, and his experiences were a great help in writing two chapters in this book. Many thanks to Sal, Sari, and Louie for supporting my ventures over the years and for sharing my excitement in this project from the onset. Thanks also to Karen Rufa for her comments and suggestions.

Thanks to my family and friends who encouraged my writing endeavor and understood my time limitations while preparing this book. Special thanks to my parents for instilling in me the notion that nothing is impossible and to Roberta Gianone for her support and advice. Thanks too to Mary Rufa, Marilyn Wilner, and David Bronstein for their expressed confidence in my abilities.

And finally, my apologies to anyone I failed to mention in this long and international roster of people who have contributed to the development of the MS-DOS Kermit program and to the publication of this book. Thank you all.

Christine M. Gianone
New York City, 1990
`cmg@columbia.edu`
`KERMIT@CUVMA.BITNET`

Preface to the Second Edition

The second edition of *Using MS-DOS Kermit* includes descriptions of the features that are new to release 3.11 of MS-DOS Kermit. These include built-in TCP/IP networking support, improved and expanded support for other networks, an easy-to-use dialing directory, support for translation of Cyrillic character sets during file transfer, a script language interface to the DOS file system, support for full-duplex RTS/CTS flow control for use with high-speed modems, user-settable serial port address and IRQ line, text and graphics terminal emulation improvements, improved printer control, longer macro definitions, and new script programming functions and variables.

The second edition also includes a new chapter on local area networks, additional material on running Kermit in windowing environments such as Microsoft Windows and Quarterdeck DesqView, a new appendix containing tables of the escape sequences used by Kermit's text and graphics terminal emulators, new character set tables, and expanded descriptions of many of Kermit's features.

Thanks once again to Professor Joe Doupnik for his continued work on the MS-DOS Kermit software and for providing most of the material in the new sections on local area networks and escape sequences. Thanks to Terry Kennedy for the LK250 keyboard driver and LK250 material in Appendix II, and to Trevor Warwick of DEC UK for adapting the

LK250 driver to the VAXmate. Thanks also to Joe Moyer, former project manager of IBM's LANACS product, for participation in the design and testing of the EBIOS network support, for recommending Kermit's inclusion with the LANACS product, and for reviewing the LANACS material in the Networks chapter.

Special thanks to Erick Engelke of Waterloo University, Ontario, for permitting his TCP/IP support code to be adapted to MS-DOS Kermit, and for his enthusiastic cooperation in this effort.

Many thanks to others who helped with the design and development of MS-DOS Kermit 3.11: Merton Crockett, John Chandler, Frank da Cruz, Max Evarts, Mike Freeman, Hirofumi Fujii, Bo Gedda, Thomas Goerz, Brian Holley, Terry Kennedy, Ted Medin, Jason Merrill, Andy Newcomb, Dan Norstedt, John Nyenhuis, Bert Tyler, Konstantin Vinogradov, Dimitri Vulis, Robert Weiner, Steve Wood, Dave Zielke, and to the hundreds of beta-testers across the world.

And finally, thanks to those who participated in the production of the second edition: the reviewers—Frank da Cruz, Joe Doupnik, Frank Dreano, Dot James, Terry Kennedy, John Klensin, Joyce Lotz, Joe Moyer, Clifford Stoll, Margaret Wilson—and the teams at Digital Press and Editorial Inc., including the new faces: production editors Marsha Finley and Mary Higgins, copy editor Dianne Wood, proof reader Jacqueline Serafino, and Monica Broadnax in Marketing.

Christine M. Gianone
New York City, 1991

Chapter 1

Introduction

Many of us are conditioned to read computer manuals only after a lot of trial and error. We are annoyed that the PC software, with all its menus and choices, wasn't self-explanatory enough to guide us. But sometimes there is simply no substitute for paper and ink. This book is here to ease the burden.

Communication software is a little more complicated than other PC-based software because *two* computers are involved instead of one. Not only are two computers involved, but these two computers must be successfully connected to each other before you can even *begin* to use a communication program. A Kermit program is even *more* complicated because it allows the other computer to be practically any make or model. But these complications allow for flexibility.

What Will Kermit Do for You?

PCs have become commonplace in both the office and the home. When PCs are able to communicate—to interact and share data—with each other, with the central mainframe computer, or with a dialup service, we can use them to provide a wider range of services. MS-DOS Kermit lets your desktop PC talk to other computers—across the office, across the city, across the continent, or across the world.

Kermit software has been used to explore outer space, to feed the hungry, and to heal the sick. It gathers data from scientific and medical devices. It transmits those amazing sports statistics to the television announcer at the game. It relays the computer commands

that control giant steel furnaces and factory-floor milling machinery. It connects the PCs and computers of most U.S. government agencies and most of the world's universities. It probably even runs in the cash register of your local fast-food restaurant.

Why is Kermit so popular? It's good and it's cheap. With Kermit, you'll be able to interact with nearby or distant time-sharing computers, large or small. You can exchange information safely and conveniently with almost any other kind of computer. You can access dialup services for news, stock quotations, electronic mail, banking, shopping, and file sharing. You can trade data with friends or neighbors, even if they have totally different kinds of computers. You can work from home. Kermit can connect you to "the network." Use your imagination. With today's worldwide telephone system, and our rapidly improving telecommunication and computing technology, there is no limit.

Capabilities of MS-DOS Kermit

MS-DOS Kermit 3.11 is one of the most advanced and powerful of all the Kermit programs. It lets you use your keyboard and screen to interact with remote computers and services over telephone lines and networks, send and receive files reliably, and automate any task you can do by hand. Here is a list of some of its features for quick reference. Don't worry if you are unfamiliar with some of the terminology. You will have a better understanding of these capabilities as you read along.

- Error-correcting, efficient file transfer using the Kermit protocol. Text or binary files may be transferred singly or in groups.
- File transfer capabilities include long packets, sliding windows, international text character set conversion, checksum and CRC error correction, eighth-bit prefixing, attribute packets, server mode, data compression, file management, and more. MS-DOS Kermit can also transmit and capture files without error checking.
- Emulation of the DEC VT52, VT100, VT102, VT220, and VT320 terminals; of the Heath/Zenith-19 terminal; and of Tektronix graphics 4010/4014 terminals. In VT220 and VT320 modes, Kermit supports a wide variety of national and international character sets.
- Operation over a variety of local area networks and protocols, including TCP/IP, 3COM BAPI, AT&T StarLAN/StarGROUP, DECnet CTERM and LAT, IBM EBIOS/LANACS, Intel OpenNET, NetBIOS, Novell NetWare NASI/NACS, Novell NetWare TELAPI, Interconnections/Novell TES, Ungermann-Bass Net/One, plus any BIOS Interrupt 14 interceptor for other network services.
- Communication settings that can be matched to virtually any other computer.

- Operation at speeds up to 57,600 bits per second (bps) or higher using a standard IBM communication port, and even greater speeds on local area networks.
- Compatibility with Microsoft Windows and similar operating environments.
- A simple, consistent, and intuitive command language with built-in help.
- Command and initialization files.
- Screen rollback, screen capture, and printer control.
- Key redefinition and keystroke macros.
- Command macros and a powerful script programming language.
- Logging, security, and debugging features.
- Special features for people who have auditory, visual, or physical challenges.

Kermit in a Nutshell

In Figure 1-1, we see a PC and a minicomputer. Each computer has something the other computer wants. The PC's disk is full of spreadsheets, databases, reports, messages, mailing lists, stories, and recipes that would be useful to the people who use the minicomputer. And the minicomputer has services the PC user needs: electronic mail, large-capacity disk storage, high-speed printers, networks, applications, and its own databases.

Figure 1-1 Two Computers in Search of a Connection

Figure 1-2 Terminal Emulation

How does the PC user send information to the minicomputer, and how does the user access the minicomputer's applications? Files cannot be shared using disks or tapes since the minicomputer does not have a diskette drive, and the PC does not have a tape drive. The answer is *data communication*—sending data between the two computers through a wire. This is Kermit's job.

MS-DOS Kermit does two things for you: *terminal emulation* and *file transfer*. Terminal emulation makes your PC act like a terminal so you can carry on a dialog with another computer. At any time during this dialog, you can send a file from your PC's disk to the other computer, or you can have the other computer send a file to your PC's disk.

Terminal emulation means that the characters (letters, numbers, and symbols) that you type on your PC's keyboard are sent through the PC's communication device to the other computer, and the characters that the other computer sends back are displayed on your screen, as shown in Figure 1-2. You have the illusion that the other computer is right in front of you. This process, called *connect mode,* allows you to use applications on the other computer, such as electronic mail, text editors, databases, programming languages, and bulletin boards, just as if you had a real terminal. For most of the 1970s, almost all computing was done in this way. Many terminals were connected to a minicomputer or mainframe, and terminal users all shared the power and resources of the central computer.

Today, terminals are an endangered species. Most computer users now have personal computers (PCs) on their desks. A PC is a general-purpose computer, capable of performing most of the central computer's tasks and many new ones. PC users can create and print documents, balance budgets, manage databases, produce sophisticated reports, write programs in many languages, and so on. But there comes a time when a PC user needs to share his or her creations with friends or co-workers, or needs access to information that is on another computer. That's where data communications and Kermit come in.

A Failure to Communicate

In data communications, Murphy's Law, "Everything that can go wrong will go wrong," is an understatement. As the computer graffiti says, "Murphy was an optimist." Let's look at what can go wrong when you try to make two computers exchange data. Kermit software takes care of these these difficulties so you won't have to worry about them.

The two computers, your PC (the *local* computer) and the computer with which you are communicating (the *remote* computer) must be physically connected to each other. The connection between your PC and the other computer can be as simple as a direct cable or as complicated as a long-distance telephone call that is routed through modems, switching stations, microwave radio transmitters, undersea cables, and earth satellites. These connections are always susceptible to interference.

On a voice telephone call, static and other types of interference are not serious problems: You can usually make out what the other person is saying despite the noise. Computers, however, are not as smart as humans. They represent and transmit data using a simple and vulnerable code consisting only of zeros and ones. Static can easily change a one to a zero or vice versa, and this will change your data. If you are sending next year's budget to the boss, you don't want the numbers corrupted by static (unless the error accidentally doubled your salary, but there are no guarantees in data communications).

The momentary silences that occur at critical parts of a telephone conversation, which blank out words or phrases, can occur in data communications too. Computers cannot necessarily tell that this has happened, and pieces of data could be missing in undetectable ways. Imagine, for example, if the word "DON'T" were deleted from the message "DON'T LAUNCH THE MISSILES NOW!"

And a phone call can go completely and undetectably dead. This can happen, for example, if the dog pulls your phone cord out of the wall during your conversation. Have you ever been talking to someone enthusiastically, on and on, when this happened? Fifteen minutes later, you pause to ask "Are you still there?" and nothing. If you call back and someone answers, maybe the lost connection was really an accident. The same thing can happen to a data connection, but usually without the dog.

Differences between the two people or computers that are communicating bring about other considerations besides the physical connection. Did you ever have someone dictating an address to you over the phone who talked faster than you could write it down? It is perfectly natural for you to say "Slow down, please," and any normal, courteous person would. But it is not so simple with computers. If a huge and powerful mainframe is sending data to a small and frail PC's diskette, the PC might not be able to record the data as fast as it arrives, and it has no built-in, natural way of saying "Slow down."

And like people, not all computers speak the same language. Different computers can use different codes for storing and transmitting data. If you get a phone call from a person who doesn't speak your language, you're probably smart enough to realize it, even if it takes a few moments to sink in. A computer that receives data in an unknown code will most likely accept it but then interpret it incorrectly. The results could be disastrous.

File Transfer: Playing by the Rules

Because so much can go wrong with a phone call, and with the people at each end, a simple but effective set of rules and procedures has evolved over the years known as telephone etiquette, and it is a good model for data communications.

First, I initiate the connection by dialing your phone number. Your phone rings, you pick it up to complete the connection, and then you say "Hello?" I say hello back and identify myself. You either agree or refuse to talk to me. If you agree, you and I take turns talking. If you're going too fast, I ask you to slow down. If there is interference and I can't understand you, I ask you to repeat yourself. When we are finished talking, we say good-bye and hang up, which breaks the connection. If either of us hangs up before saying good-bye, it is a breach of etiquette.

When a set of rules and procedures grows sufficiently complicated and formal, it becomes a *protocol*. We are familiar with this term from diplomacy, a sphere in which people who might not share the same native language or customs must nevertheless communicate with clarity and precision.

Data communications protocols are even greater sticklers for detail than diplomats. They are how computers of different sizes, shapes, speeds, operating systems, and codes and from different manufacturers and countries can meaningfully communicate. This is done by exchanging strictly formatted messages according to well-defined and agreed-on rules.

But computers cannot do this on their own. People must formulate the rules and write computer programs like Kermit to carry them out. Kermit's rules for file transfer, illustrated in Figure 1-3, are similar to telephone etiquette:

This can be the same as Figure 9-1

Figure 1-3 Kermit to Kermit

1. Make the initial connection, say hello, and make sure the other computer is ready to transfer a file.
2. Tell the other computer the name of the file it will be getting and (if the other computer is interested) the file's size, creation date, and other particulars.
3. Break the file up into smaller pieces called *packets*, and send one packet at a time.
4. The file receiver inspects each arriving packet for damages and says "Yes" to accept it or "No" to reject it.
5. When a packet is accepted, the next one is sent. If a packet is rejected, it is sent again.
6. If the same packet is rejected too many times, or if an acknowledgment never arrives, the file transfer fails.
7. When the file has been completely transferred, the receiver is told "End of File!"
8. If there are more files to send, steps 2 through 7 are repeated for each file.
9. When no more files remain to be sent, the two Kermits say goodbye to each other.

Two telephones cannot communicate with each other by themselves. They need a pair of people to operate them, and conversations will not be successful unless the two people follow the same set of rules. Likewise, two computers cannot transmit data using the Kermit protocol unless a Kermit program is running on *each* computer at the same time. What makes Kermit different from a regular phone call is that (in most cases) *you* are controlling both ends of the connection yourself.

But don't let this frighten you. Controlling two computers simultaneously is much easier than, say, driving two cars at once. It can even give you a sense of power. The next few chapters will get you started.[2]

[2]If you're interested in the details of the Kermit protocol, read *Kermit, A File Transfer Protocol* by Frank da Cruz, Digital Press, 1987.

Chapter 2

Getting Started

The MS-DOS Kermit diskette contains the executable Kermit program for the IBM PC and compatibles, `KERMIT.EXE`, along with several other important files that are described in Appendix III.

To use the MS-DOS Kermit software to communicate with another computer, you'll need a PC, the software that comes with this book, enough disk space to hold any files you want to transfer, and a way to communicate with the other computer:

- An IBM PC or compatible. Examples include: An IBM PC, PC*jr*, Portable PC, PC/XT, PC/AT, PC Convertible, PS/1, PS/2, or any true IBM compatible such as Compaq, DEC VAXmate or DECstation PC, Dell, Northgate, Toshiba (US models), Zeos, etc.[3]
- The MS-DOS or PC-DOS operating system software for your PC, version 2.0 or later.
- A 3.5-inch or 5.25-inch diskette drive.
- The Kermit software on a diskette, which comes with this book.

---

[3]Special versions of MS-DOS Kermit are also available for non-IBM-compatible PCs, including the DEC Rainbow, Victor 9000, HP-150 and Portable Plus, Grid Compass, and many others, including a Japanese version for the NEC PC9801 that handles Japanese character sets.

- A blank diskette or a hard disk drive with at least 360K[4] bytes of available space.
- 220K bytes of available memory. Kermit will use more if it is available, and can also be configured to use less.
- At least one of the following communication devices:

  A serial port (asynchronous adapter) installed in your PC and, if you will be making data calls over a telephone, an external modem.

  An internal modem.

  A network adapter with appropriate driver software.
- A cable that connects either:

  Your PC's serial port to an external modem (a *modem cable*).

  Your PC's serial port to another computer (a *null modem cable*).

  Your internal modem to the telephone line (a *telephone cable*).

  Your network adapter to the network (a *network cable*).
- For Tektronix graphics terminal emulation, any of the common graphics adapters (CGA, EGA, VGA, XGA, Hercules, AT&T/Olivetti, and so forth).

Installation

MS-DOS Kermit may be run from a diskette or a hard disk. In either case, the very first thing you should do is make a working copy of the Kermit diskette from the original, and then put the original away for safekeeping. Since Kermit programs are not copy-protected, you can make as many copies as you like, give them away to your friends and colleagues, and contribute the program to your local user group for further distribution.

To install Kermit, follow the steps that apply to your PC's configuration. If you also want to configure Kermit for use under Microsoft Windows, Quarterdeck DesqView, or OS/2, see page 203.

If you have trouble understanding any of the instructions in this chapter, please skip ahead to the next chapter, "Basics of MS-DOS," and then come back to this chapter and try again.

---

[4]K means kilo, or one thousand. 360K bytes is approximately 360,000 characters.

Figure 2-1 Installing Kermit in a Single-Diskette PC

Single Diskette Systems

If your PC has only one diskette drive, like the one in Figure 2-1, follow these steps to copy your Kermit diskette. `A>` is the DOS[5] prompt that tells you DOS is ready for you to give commands to it. You type only the commands that are underlined, and you must terminate each command by pressing the Enter key.

1. Make sure your DOS diskette is in drive A.
2. Start up your PC (if it isn't already started). You should see the DOS `A>` prompt.
3. Make sure the write-protect notch on the Kermit diskette is covered so that you don't accidentally write over it. This is the *source diskette*.
4. Get a blank diskette and make sure the write-protect notch is uncovered so that you can copy the Kermit files to it. This is the *target diskette*.
5. Type `diskcopy a: a:`, and then press the Enter key.
6. A message will appear asking you to insert the source diskette into drive A. Remove the DOS diskette and put it safely out of reach. Put the original Kermit diskette into drive A (see Figure 2-1), and then press the Enter key.
7. After the DISKCOPY command has read as much of the source diskette as it can fit into its memory, it will ask you to remove the source diskette (original Kermit disk) from drive A and insert the target diskette (blank diskette). Do this, and then press the Enter key.
8. Repeat steps 6 and 7 until the Kermit diskette is copied. A message will ask you if you want to copy another diskette. Press N for NO.

---

[5]DOS is short for MS-DOS or PC-DOS.

Figure 2-2 Installing Kermit in a Dual-Diskette PC

9. Remove the new Kermit diskette, and then label it.
10. Put your new Kermit diskette into drive A.
11. At the DOS `A>` prompt, type `kermit`, and then press the Enter key. You should see the `MS-Kermit>` prompt. If not, review the previous steps.
12. At the `MS-Kermit>` prompt, type `exit`, and then press the Enter key to exit from the Kermit program.

Dual Diskette Drive Systems

On a system with two diskette drives, follow these steps. `A>` is the DOS prompt that tells you DOS is ready for you to give commands to it. You type only the commands that are underlined. Remember, you must terminate each command by pressing the Enter key.

1. Make sure your DOS diskette is in drive A (the upper or left diskette drive).
2. Start up your PC (if it isn't already started). You should see the DOS `A>` prompt.
3. Make sure the write-protect notch on the Kermit diskette is covered so that you don't erase it by mistake. This is the *source diskette*.
4. Get a blank diskette and make sure the write-protect notch is uncovered so that you can copy the Kermit files to it. This is the *target diskette*.
5. Type `diskcopy a: b:`, and then press the Enter key.
6. A message will appear asking you to insert the source diskette into drive A and the target diskette into drive B (lower or right diskette drive). Remove the DOS diskette, and put it safely out of reach. Put the original Kermit diskette into drive A and the blank diskette into drive B (Figure 2-2), and then press the Enter key.

Figure 2-3 Installing Kermit in a PC with a Hard Disk Drive

7. When the diskette is copied, a message will ask you if you want to copy another diskette. Press N for NO.
8. Remove the new Kermit diskette, and then label it. Remove the original Kermit source diskette.
9. Put your new Kermit diskette back into drive B and the DOS diskette into drive A.
10. At the DOS `A>` prompt, type `b:kermit`, and then press the Enter key. You should see the `MS-Kermit>` prompt (if not, review the previous steps). To return to DOS, type `exit` and then press Enter.

Hard Disk Drive Systems

For easy access, you should copy all the files from the original Kermit diskette to your hard disk. You will need 360K available space on your hard disk. Follow these instructions. You type what is underlined. Remember, you must terminate each command by pressing the Enter key.

1. Start up your PC (if it isn't already started). You should see the DOS hard disk prompt, which is usually `C>`.
2. Check whether you have enough space on your hard disk. Give the following DOS command (the example assumes `C>` is the prompt):

   ```
   C>dir | find "bytes free"
   ```

   If there are fewer than 360,000 bytes free, you must delete some files to make room for Kermit.
3. Make sure the write-protect notch on the Kermit diskette is covered so that you don't destroy your Kermit files inadvertently.
4. Put the Kermit diskette into drive A (upper or left diskette drive). (See Figure 2-3.)

5. Make a KERMIT directory on your hard disk to hold the Kermit files, and then make it your default directory. In the following example, we assume that your hard disk is disk C. Other disk letters and directory names can be used. The DOS prompt is C>, and the commands that follow it are the ones you should type. Terminate each command by pressing the Enter key:

   ```
   C>mkdir \kermit
   C>cd \kermit
   ```

6. At the DOS prompt, which might have changed to C:\KERMIT> after you gave the CD command (see page 25), type:

   ```
   C>copy a:*.*
   ```

   and then press the Enter key.

7. The files are now in the KERMIT directory on your hard disk.

8. Remove the Kermit source disk from drive A.

9. At the DOS prompt, type:

   ```
   C>kermit
   ```

   Remember to press the Enter key after you type kermit. You should see the MS-Kermit> prompt. If not, review the previous steps.

10. At the MS-Kermit> prompt, type exit to leave the Kermit program and return to DOS.

Making Sure DOS Can Find Kermit

The KERMIT directory, which now contains the MS-DOS Kermit program, KERMIT.EXE, should be added to your DOS search path. This will let you run the program simply by typing the word kermit, no matter what your current disk and directory are. The other files on the original Kermit diskette should also be stored in the KERMIT directory so Kermit can find them.

To find out what your DOS search path is, type the command path at the DOS prompt. The response will be either:

```
No Path
```

or something like:

```
C:\;C:\BIN
```

In the second case, the device–directory combinations are separated by semicolons. If you have no path, you should make one by using a text editor to add a PATH= command to the AUTOEXEC.BAT file in the top-level directory of your DOS startup disk, for example:

```
PATH=C:\;C:\KERMIT
```

If you do not know how to use a text editor, see the section "Creating and Modifying Files" in Chapter 3 for instructions.

If you already have a path, you should add the `KERMIT` directory to it. Assuming that your current path is `C:\;C:\BIN;` as in the example above, use a text editor to change the line:

```
PATH=C:\;C:\BIN
```

in your `AUTOEXEC.BAT` file to:

```
PATH=C:\;C:\BIN;C:\KERMIT
```

This means that whenever you type a command that is not built into DOS, and it is not the name of a program stored in your current directory, DOS is to look on the C disk, first in the top-level directory, then in the `\BIN` directory, and then in the `\KERMIT` directory, for a program of that name and run it.

If you used a word processor such as Microsoft Word to edit your `AUTOEXEC.BAT` file, be sure to save the file in "text-only" mode (see page 25). To get your new PATH= statement to take effect you have to restart your PC. From now on, you can run Kermit just by typing its name, `kermit`.

Chapter 3

Basics of MS-DOS

I am sure you are anxious to begin using the Kermit software, but if you are not familiar with the filenames, file formats, directories, devices, and commands used by your PC, you should read this chapter first. If you are an experienced PC user, please feel free to skip ahead to Chapter 4.

MS-DOS means Microsoft Disk Operating System. This is the program that controls your PC and all its devices. In this book, the words DOS, MS-DOS, and PC-DOS are used interchangeably.

Using MS-DOS

Your DOS session begins when you turn on your PC. Your startup disk—either your hard disk or a diskette you have put into one of your diskette drives—must contain the DOS operating system and a copy of the `COMMAND.COM` file. `COMMAND.COM` is the program that reads your commands from the keyboard and executes them.

When DOS is active, a "prompt" appears on the screen, usually the letter of the current disk, followed by a right angle bracket, for example:

```
C>
```

The cursor (normally a blinking underline) sits to the right of the prompt, waiting for you to type a command, indicating your current position on the screen. To execute a DOS command, type the command's name and then press the Enter key:

```
C>ver
IBM Personal Computer DOS Version 3.30
C>
```

Letters can be entered in either upper or lower case. In this example, you have asked DOS to display its version number by typing the VER command. The underlining shows the letters that you type. When the command has finished its work, a new prompt appears. If you give a command that DOS does not understand, or cannot find, its response is:

```
Bad command or file name
```

While typing DOS commands, you can correct mistakes by pressing the Backspace key to erase single characters, or the Esc key to erase the whole command. You can edit commands only before you have pressed the Enter key.

If you want to stop the execution of a command after it has been entered, type "Control-C," that is, hold down the Ctrl key and press the letter C.

If you need to restart DOS, hold down the Ctrl, Alt, and Del keys simultaneously. If that doesn't work, turn your PC off and then back on again. Restarting DOS is called "rebooting." Rebooting is your last resort when a DOS command or application is hopelessly stuck.

DOS Filenames

A DOS filename has two parts, the *name* proper and the *type*. The two parts are separated by a period. For example, every DOS computer has a file called `COMMAND.COM`, which is the DOS command processor. The name is `COMMAND`, and the type is `COM`. Names can be up to eight characters long, and types can be up to three characters long. Legal characters include letters, digits, and certain punctuation marks. Letters are uppercase. If you type a lowercase letter in a filename, DOS converts it to uppercase. Some typical filenames are `AUTOEXEC.BAT`, `CONFIG.SYS`, and `KERMIT.EXE`. DOS filetypes, by convention, tell the purpose of the file, for example:

`.BAT` A DOS batch file, containing DOS commands to be run.

`.COM` A machine-language program to be run.

`.EXE` A machine-language program to be run.

`.DOC` Documentation, text to be printed or displayed on the screen.

`.TXT` Text to be printed or displayed on the screen.

`.HLP` A help file to be printed or displayed on the screen.

`.WKS` A Lotus 1-2-3 spreadsheet file.

`.DBF` A dBase database file.

`.PIF` A Microsoft Windows Program Information File.

`.BAS` A BASIC program.

`.C` A C-language source program.

There is no universal standard for DOS filetypes, except that DOS will run only programs with filetypes of `.BAT`, `.EXE`, or `.COM`.

To display a file with any of the above filetypes on the screen, use the DOS TYPE command, for example:

```
C>type autoexec.bat
```

If the result is comprehensible and sensibly formatted, the file is probably a *text file*. On the other hand, if the result is "garbage"—a lot of funny-looking characters and sound effects—it is probably a *binary file*, which is meant not for viewing by humans but for processing directly by the computer. For example, the binary file `COMMAND.COM` is a PC machine-language program.

DOS Directories

DOS lets you organize your files into separate areas called *directories*. If you think of your disk as a filing cabinet, a directory is like one of its drawers. If you were a compulsive filer and each of your file drawers had its own index, the analogy would be complete.

Directories allow you to collect related files together. Each file within a directory must have a unique name, but files that are in different directories can have identical names. A directory name in DOS looks similar to a filename, but it begins with a backslash (\) character, for example:

```
\PROGRAMS
```

An important notion in DOS is the *current directory* .You can think of this as the file drawer that's open. To refer to a file in your current directory, just type its name. To refer to a file in a different directory, you have to include the directory name. A backslash must separate the directory name from the filename:

```
\PROGRAMS\OOFA.C
```

The *top-level* directory of any disk is simply `\`. Directories beneath it are called *subdirectories*. In the example above, `\PROGRAMS` is a subdirectory of the top-level directory, and `OOFA.C` is a file in that subdirectory.

To find out what your current directory is, just type CD at the DOS prompt, and then press the Enter key:

```
C>cd                              (Type the CD command)
C:\                               (See the result)
C>                                (Next DOS prompt)
```

This shows that your current directory is the top-level directory on the C disk.

To change your current directory, use the DOS command CD to specify the directory you want and then press the Enter key, for example:

```
C>cd \programs                    (Change directory)
C>cd                              (Check it)
C:\PROGRAMS                       (See, it worked)
C>
```

You can set up your DOS system to include the current disk and directory name in your DOS prompt so you can always tell where you are. To do this, give the following command:

```
C>prompt $p$g                     (Old prompt is C>)
C:\>                              (New prompt shows directory)
C:\>cd \programs                  (Change directory)
C:\PROGRAMS>                      (Prompt shows new directory)
```

Subdirectories can have their own subdirectories. For example, the PROGRAMS subdirectory might contain a subdirectory called NEW:

```
C:\PROGRAMS\NEW
```

If it doesn't, you can create it. To create a subdirectory, use the DOS MKDIR command:

```
C>mkdir \programs\new
```

You can use the RMDIR command to remove a directory (but only if it has no files in it).

You can move directly to a subdirectory within another subdirectory by telling DOS the full subdirectory name:

```
C>cd \programs\new
C:\PROGRAMS\NEW>
```

If you omit the initial backslash, the directory name is taken to be a subdirectory of the current directory. This is called a *relative path*. For example, if your current directory is \PROGRAMS, you can refer to a file in its NEW subdirectory as:

```
NEW\MYFILE.TXT
```

Two special directory names are related to your current directory: . (one period) means the current directory, and .. (two periods) means the directory immediately "above" the

current directory. You can use these names in the CD command and in filenames. For example, if you were in the subdirectory \NEW\ and you wanted to get back to \PROGRAMS, the directory above it:

```
C\PROGRAMS\NEW>cd ..
C\PROGRAMS>
```

DOS Devices

Just as each directory may contain many files, each disk device may contain many directories. DOS disks are identified by letters: A, B, C, etc. Usually the A and B disks are diskettes, and the C disk, if any, is a hard disk. The disk letter may be included in a file specification to refer to a file that is on a disk different from the one your current directory is on:

```
A:\PROGRAMS\OOFA.C
```

Each disk has its own current directory, and you may switch among disks and their current directories while at the DOS prompt by just typing the disk letter followed by a colon and then pressing the enter key:

```
C>a:                          (Switch to disk A)
A>                            (Current disk is now A)
```

DOS also has nondisk devices, including the printer, the communication port, and the console. These are referred to by a simple name—no colon (as in the disk device name) and no dot (as in a filename). For example, COM1 is communication port 1, PRN is the printer, and CON is the console. These device names may be used in DOS commands in place of filenames, sometimes with bizarre results. A particularly useful device is the *null device*, NUL. Output sent to this device disappears mysteriously but with no ill effect. For input, NUL acts like an empty file. These device names may differ on non-IBM-compatible DOS systems.

Running DOS Commands

DOS has two kinds of commands: *internal* and *external*. An internal command is built into DOS, and you can always run it by typing its name in response to the DOS prompt, for example, the TYPE command:

```
C>type oofa.c
```

An external command is a file with a filetype of .EXE, .COM, or .BAT. You can run it by simply typing its name (either with or without the filetype) *if* it is in the current directory on the current disk. For example, if the file is called OOFA.EXE, you can type:

```
C>oofa
```

If it is on a different disk or directory, you must include the disk and/or directory when you type the command:[6]

```
A>C:\programs\oofa
```

or if the disk–directory combination is in your current PATH, you can just type its name:

```
A>path                          (What's the current path?)
PATH=C:\;C:\PROGRAMS            (C:\PROGRAMS is in it)
A>oofa                          (so I can just type OOFA)
```

It is normal practice to have the files that came on your DOS diskette always mounted in your A drive if you don't have a hard disk. If you have a hard disk, you should have copied the files from your DOS diskette into a directory on the hard disk. Let's say you have copied them to the top-level directory on the C disk. To make sure all of your DOS commands are in your path, you should have a line like this in your AUTOEXEC.BAT file:

```
PATH=C:\
```

The AUTOEXEC.BAT file is executed automatically by DOS when you start your PC. It must be in the top-level directory of the disk that you start DOS from.

For a diskette-only system, for instance one in which you have the DOS diskette in drive A and your working diskette in drive B, the PATH statement might look like:

```
PATH=A:\;B:\
```

If you set up your PATH this way, make sure you always have a diskette in each drive or the PC will get stuck.

Your PATH can include a list of disks and directories that DOS will search when you type a command name:

```
PATH=C:\;C:\PROGRAMS;F:\BIN
```

DOS searches these areas from left to right. So if you have two files named OOFA.EXE, one in C:\PROGRAMS and one in F:\BIN, and you type oofa, DOS will run the version it finds first, that is, C:\PROGRAMS\OOFA.EXE.

As you can see, the PATH mechanism of DOS allows you to add new commands simply by storing programs or batch files in any directory in your DOS search path.

---

[6]DOS 3.0 or later. Earlier DOS versions do not accept a directory name.

Commonly Used DOS Commands

The most commonly used DOS commands are for *file management:* displaying, printing, copying, renaming, and deleting files. DOS commands like TYPE and DIR, which display text on your screen, can be controlled using the following *control characters*. Hold down the Ctrl key, and then type the indicated letter, but do *not* press the Enter key:

`Ctrl-C` Interrupts the command and returns you to the DOS prompt.

`Ctrl-S` Stops the display on the screen so that you can read it.

`Ctrl-Q` Resumes the screen display.

Here is a brief summary of file-related DOS commands. You must press the Enter key when you are finished typing the command and are ready for it to be executed. Refer to your DOS manual for details.

TYPE *filename*

Displays the file on your screen, for example:

```
C>type oofa.hlp
```

Try this: TYPE a long file, such as `KERMIT.HLP` on your Kermit disk. Use `Ctrl-S` (hold down the Ctrl key and press the S key) to stop the display and `Ctrl-Q` (hold down the Ctrl key and press the Q key) to resume the display. Use `Ctrl-C` (hold down the Ctrl key and press the C key) to cancel the TYPE command.

DIR

Lists the names, sizes, and creation dates of all the files in the current directory. If you include a disk letter (and the colon), directory name, or filename, the files in the specified area, or with the specified name, are listed:

```
C>dir
C>dir a:
C>dir a:\programs
```

CHKDSK

Tells how much space is left on the current disk. If you include a disk letter (and the colon), the specified disk is checked:

```
C>chkdsk c:
```

PRINT *filename*

Prints the file on your printer, if you have one:

```
C>print oofa.c
```

COPY *filename1 filename2*

Creates the second file, which is a copy of the first file. The files may be on different devices and directories:

```
C>copy oofa.c a:\programs\new
```

If *filename2* is omitted, or is specified as `.` (period), the new files are created on the current disk and directory with the same names as the original files:

```
C>copy a:*.*
```

REN *filename1 filename2*

Changes the name of *filename1* to *filename2*

```
C>ren oofa.c serious.c
```

This command cannot be used to move a file to another device or directory.

DEL *filename*

Deletes (removes, erases) the file:

```
C>del oofa.exe
```

Wildcards

Certain DOS commands, like DIR, COPY, PRINT, and DEL, can operate on more than one file at a time. To specify a group of files, you can use *wildcard* characters in the filename (but not a directory or device name). DOS recognizes the following two wildcard characters:

`*` (asterisk) Matches all characters from the current position to the end of the current field (filename or filetype).

`?` (question mark) Matches a single character in the current position.

So `O*.HLP` would match any file (like `OOFA.HLP`) whose name started with the letter `O` and that had a filetype of `.HLP`. The names could be of different lengths, like `O.HLP`, `OK.HLP`, `OLIVE.HLP`, and `OOMPAH.HLP`. If all of these files existed in your current directory, the command:

```
C>del o*.hlp
```

would delete all of them. The specification `*.*` matches all the files in the current directory. If you tell DOS to:

```
C>del *.*
```

DOS will respond:

```
Are you sure (Y/N)?
```

This gives you a chance to change your mind. Combining wildcard characters can result in more specific selections; for example:

```
O?F*.?O*
```

would match all files whose names start with the letter O, followed by any single character, followed by the letter F, followed by zero or more additional characters, with a filetype two or three letters long whose second letter is O. This would match files with names like OOFA.COM, OAF.DOC, and OFFICE.LOG. Although this method may seem cumbersome, it is the only method DOS provides for identifying many files all at once. Once you become used to wildcards, they can save you a lot of time and typing.

Creating and Modifying Files

To create a new file, or to modify an existing one, you should use a *text editor* or *word processor*. DOS includes a very simple text editor called EDLIN, which is described in the DOS manual. There are many other text editors available, including commercial software programs that include a large selection of powerful features.

Let's use EDLIN to add two lines to your AUTOEXEC.BAT file: one to set your DOS prompt to show your current disk and directory, and another to set up your DOS path. For simplicity, we will add them to the end of the file, but you could also put them elsewhere in the file:

```
C>edlin \autoexec.bat
End of input file
*i#                              (Insert lines at end)
     12: PATH C:\;C:\KERMIT      (Add PATH command)
     13: PROMPT $P$G             (Add PROMPT command)
     14: Ctrl-C                  (Hold down the Ctrl key and)
                                 (press the letter C)
                                 (to leave insert mode)
*e                               (Exit and save the file)
C>
```

This works even if you don't already have an AUTOEXEC.BAT file. EDLIN will create one for you. The PATH command shown above is only a sample. Replace it with one appropriate to the organization of your disks and directories.

If you are using a word processing program to create or modify a DOS or Kermit command file, do not include any special effects (bold, underline, italics), and be sure to save the file in *text mode*. The method for doing this depends on the word processor.

In Microsoft Word 5.0, for example, press the Esc key to get to the menu, press T to choose Transfer, press S to choose Save, type the filename, use the arrow keys to get to the "format" line, choose Text-Only, press Enter to save the file, and then leave the program by pressing the Esc key and then Q.

MS-DOS Quick Command Reference

Here is a list of handy DOS commands for Kermit users. Substitute real filenames, directory names, etc., for the items shown in italics. See your DOS manual for details on these commands, as well as for more advanced topics such as batch programming and the MODE command.

CD	Display current directory.
CD *directory*	Change current directory.
CHKDSK	Check space on current disk.
CLS	Clear the screen.
COMP *file1 file2*	Compare files.
COPY *file1 file2*	Copy files.
CTTY *device*	Change console.
DATE	Display or set the current date.
DEL *file(s)*	Delete file(s).
DIR *file(s)*	List files.
EDLIN *file*	Create or edit a file.
FIND "*text*" *file*	Find text in a file.
FORMAT *disk*	Format a diskette.
MKDIR *directory*	Create a directory.
MORE < *file*	Display a file, one screen at a time.
PATH	Display PATH.
PATH=*list*	Set PATH to list of directories.
PRINT *file*	Print a file.
PROMPT *text*	Set system prompt.
REN *file1 file2*	Rename a file.
RMDIR *directory*	Remove a directory.
SET	Display environment variables.
SET *name=value*	Define an environment variable.
TIME	Display or set the current time.
TYPE *file*	Display a file.
VER	Display DOS version number.

Chapter 4

Cables, Connectors, and Modems

Your PC cannot communicate with another computer until you have established a physical connection (see Figure 4-1). If you already have a data connection to the other computer, skip ahead to Chapter 5 to check that it is working properly. There are several methods you can use to connect your PC to the outside world, depending on the communication hardware you have. If your PC does not have any kind of communication hardware, you must decide what to buy. Kermit lets you use your serial port, built-in modem, or a network adapter. This chapter and the next one discuss serial port and modem connections. Network connections are covered in Chapter 16.

Figure 4-1 A Connection Waiting to Happen

Figure 4-2 Computers Connected by Telephone

Just as there are many kinds of computers you can connect to, there are many ways to make the physical connection. Once you establish the best way to set up your PC connection, however, you will probably not need to alter it. So, think of this chapter as a one-shot deal and it should seem less painful.

Modems

If you will be using your PC to make data connections over the telephone system, you will need a *modem*, which is a device that allows computer data to be transmitted over ordinary telephone lines by converting between digital computer signals and analog telephone signals. To communicate in this manner, *both* computers must have a modem—one to convert to analog signals so the data can travel over telephone wires and one to convert back to digital signals so the computer can understand the data (see Figure 4-2.)

Unless both communicating computers belong to you, you have to be concerned only with the modem for your own PC. Bulletin boards and other dialup services already have modems on their end that are set up for incoming calls.

You can choose from two types of modems: *external* and *internal*. An external modem is a separate unit that lies between your PC's serial port and the telephone wall jack. An internal modem is installed inside your PC and connects directly to the telephone wall jack.

Although an internal modem is less expensive and does not take up any space on your desk, it is also less flexible. An external modem can be used on different types of computers and is easily mobile. An internal modem can usually be used on only one type of computer and is physically installed inside it. Kermit works with any external modem, but Kermit does not necessarily support all makes and models of internal modems. Therefore, if you must buy a modem, the external type is recommended.

Locating and Identifying Your Communication Device

Before you can use your PC for communications, you must have an *internal modem* or a *serial port*. These are circuit boards installed in your PC that have a connector for a cable. Look at the back of your PC. You will see a number of connectors for various devices. Unfortunately, these connectors are rarely labeled so finding your communication device might require a little detective work.

Internal Modems

If your PC has an internal modem, you will see a modular telephone jack somewhere on the back of your PC. This telephone jack allows the modem to be connected to your telephone wall jack using a modular phone cable (available in hardware stores and supermarkets).

The internal modem can be used only to communicate with another modem. It cannot be connected to another computer's serial port or to other data communication equipment like terminal servers and multiplexers.

Most internal modems are available for use whenever the PC is turned on, but certain portable or laptop PCs require their internal modems to be turned on in order to conserve battery power. This might be done with a DOS command like MODE MODEM ON or MACHINE MODEM ON. In some cases there is an ON/OFF switch. Consult your owner's manual.

Serial Ports

The serial port, also called the *asynchronous adapter* or *communication port*, can be used to communicate directly with another nearby computer, or it can be used with an external modem to communicate with more distant computers or services that can be dialed by telephone. It can also be used with other kinds of data communication equipment, including terminal servers, multiplexers, and PBX data phones.

A PC serial port connector is shaped like an elongated capital letter D. For this reason, these connectors are called *D-connectors*. D-connectors come in two "genders," female and male. A female connector has holes, and a male connector has pins. This is similar to an electrical outlet (female) and plug (male).

D-connectors for serial ports also come in two sizes. One size has 25 holes, or pins, in two rows: 13 in one row and 12 in the other. This size is called DB-25. A male DB-25 connector may have fewer than 25 pins, but it always has 25 positions for pins. The female DB-25 always has 25 holes. The other size is called DB-9. It has 9 holes or pins in two rows: 5 in one row and 4 in the other.

The PC serial port uses a *male* DB-25 or DB-9 connector. Female connectors are used for other devices like printers and cannot be used for communications. Be careful not to mistake a parallel printer port for a serial communication port; damage can result!

In general, you will find DB-25 serial port connectors on IBM PCs and PC/XTs, and DB-9 connectors on PC/ATs. On the PC/AT, the DB-9 is on the same card (called the Serial/Parallel Adapter) with a DB-25 female printer port.

The PS/2 has a DB-25 as its first communication port, but additional ports, if present, have DB-9 connectors.

Serial Port Cables

To communicate, you must connect your serial port to another device with a data cable. This cable contains wires that carry your data and other signals between your PC and the other device (see Table I-1 on page 269 for a list of the signal assignments). The end that you attach to your port must be the same size and shape, DB-25 or DB-9, as the port itself but of the opposite gender. Since the port connector on the PC is male, the cable connector must be female.

The connector on the other end of your cable depends on what you will be plugging it into. In most cases, this will be a data communication device such as an external modem. Almost all such devices have *female* DB-25 connectors, so the far end of your cable would need a *male* DB-25 connector.

A cable that connects your PC to a data communication device is called a *modem cable*. Modem cables for PCs, PC/ATs, and PS/2s are readily available in computer stores and supply catalogs.

If you have a serial port but do not have a cable for it, you should buy a modem cable. Be sure to specify the size and gender of both the serial port connector and the connector on the device you will be connecting your PC to. If you will be connecting your PC directly to another computer, you will also need an adapter called a *modem eliminator*, which is explained later.

Figure 4-3 An External Modem Connection

Connecting Your PC to an External Modem

External modems are probably the most common way for PC users to connect with other computers. Most external modems have a 25-pin female connector, which must be connected to your PC's serial port with a modem cable. Be sure you are using a modem cable, *not* a printer cable or a *null modem* cable. Be sure also to make a firm connection at both ends, even if that means using the screws.

If your modem is a direct-connect model, you must also use a modular telephone cable to connect it to the telephone jack where your telephone normally plugs in, as shown in Figure 4-3. Most direct-connect modems also let you plug your telephone into the modem so the phone can be used for voice calls without disconnecting the modem (but not at the same time that a data call is in progress).

If your modem has an acoustic coupler (a rare item today), you won't need a modular telephone cable. Just push the telephone handset's earpiece and mouthpiece firmly into the modem's rubber cups, as shown in Figure 4-4.

Here is the normal procedure for connecting your PC to a direct-connect external modem (see your modem manual for more specific details):

1. If your telephone does not have a modular jack, buy a telephone that has one, and convert your wall jack to a modular one. The parts are available in any hardware store.

Figure 4-4 An Acoustically Coupled Modem Connection

2. If the modem has a power switch, turn it off. Connect the power cable and transformer, if any, to the modem and to an electrical outlet.
3. Connect the modem to the PC using your modem cable.
4. Disconnect the modular telephone cable from your telephone, and plug it into the modular phone jack on the modem. If there are two such jacks, use the one marked LINE or TO LINE.
5. If your modem has two modular jacks, use the short modular telephone cable that came with the modem to connect it to the telephone. One end goes into the modem jack marked PHONE or TELSET, and the other end goes into the empty jack on your telephone from the previous step.

Connecting the modem to your phone is not necessary for data communication. It just lets you use your phone for regular voice calls without having to unplug all the cables you have just connected. When the modem is turned off, you can use your phone normally.

The danger here is if other people have access to the telephone line you are using. For example, if you are dialing Dow Jones to find out how your stocks are doing and someone picks up an extension, your data connection will be lost. The connection can also break if you have "call waiting," so be sure to inform potential callers in advance when you want

Figure 4-5 Connection to a PBX

to use the phone for data or (if your telephone service provides this option) turn off call waiting for the duration of your modem call.

PBX Data Lines and Other Communications Equipment

Many large organizations have their own internal telephone systems called PBXs (Private Branch Exchanges). Some of these can accommodate both voice and data calls on the same telephone line simultaneously. Connection to a PBX data telephone is very similar to connecting to an external modem. Typically, the telephone will have a 25-pin female data connector just like an external modem. However, in this case, you connect your PC directly to the telephone's data connector using a modem cable, as shown in Figure 4-5, but you should leave the telephone cables alone. (See your PBX telephone manual for further information.)

Other data communication devices, including port contention units, terminal servers, and multiplexers, are generally just boxes that have one or more modemlike 25-pin female connectors, as in Figure 4-6. In this case, you can connect your PC directly to an unoccupied connector on the data communications device using a modem cable. (See the manual for the specific data communications device for details.)

Figure 4-6 Connection to a Multiplexer or Terminal Server

Figure 4-7 Internal Modem Connection

Connecting an Internal Modem to the Telephone Line

An internal modem can be used only with a modular telephone. Connect your internal modem to your telephone's wall jack. To do this, unplug the modular cable from your telephone and plug it into your internal modem's modular jack on the back of your PC. This jack is usually marked LINE. If there are two jacks on your internal modem, read your modem manual and follow the instructions for how to use them. Figure 4-7 shows a possible setup.

Figure 4-8 Direct Connection, Computer to Computer

Connecting Your PC Directly to Another Computer

If you will be connecting your PC directly to another nearby computer (one, that is, within about 50 feet or 15 meters), your PC must have a serial port. Certain signals must be crossed when connecting a computer to another computer instead of to a modem. The recommended method is to use a regular modem cable, just as you would use with an

Figure 4-8 (continued) Direct Connection, Computer to Computer

external modem, in conjunction with a *null modem adapter* or *modem eliminator*. This is a small, squarish, solid adapter plug, with a 25-position connector on each end. Internally, the pins and holes are interconnected in mysterious ways you needn't bother about. Modem eliminators are available in computer stores and catalogs, and come in three possible *gender* configurations: female–male, female–female, and male–male. Figure 4-8 shows a direct computer-to-computer connection.

Your choice of null modem depends on which kind of connector the other computer has:

- 25-pin male connector: Connect your modem cable to the other computer through a female–female null modem.
- 25-hole female connector: (Make sure this is not a parallel printer port.) Connect your modem cable to the other computer through a female–male null modem.
- Anything else at all: Use the modem cable supplied with the other computer to connect it to your PC's modem cable, with a female–female null modem in between.

For example, suppose you want to connect a PC/AT with a Macintosh II. Neither of these systems has a standard 25-pin connector. The PC/AT has a 9-pin connector, and the Macintosh has an 8-pin connector. You will probably not be able to find a null modem cable in any store that will connect these two computers. But it's easy to find *modem* cables for both the PC/AT and the Macintosh. Now all you need is a female–female modem eliminator between the two modem cables, and you're all set.

Chapter 5

Testing the Connection

If you have never used your communication hardware, cable, or modem before, you should test it now. Otherwise you might blame the innocent MS-DOS Kermit program for any failure to communicate.

Direct Connections

Read this section if you have two computers connected with only a cable—no modems involved. There are two cases: PC to PC and PC to host computer.

PC to PC

The two PCs should have a null modem connection (see Figure 5-1). You can perform this test by running back and forth between the two PCs, or you can have a friend handle the PC at the other end.

To test the connection, run Kermit on both computers, setting the programs to the same transmission speed and connecting them with the proper Kermit command. When you type characters (letters, numbers, and keyboard symbols) on one of the PC keyboards, they should appear on the other PC's screen. This should work in both directions.

Follow this example. For simplicity, let's assume you have your Kermit diskette in the A drive. In the example, the parts you type are underlined and should be terminated by pressing the Enter key, except where you see `Alt-X` (which means hold down the Alt key and press the X key).

Figure 5-1 Testing a PC-to-PC Connection

PC Number One	*PC Number Two*
A>kermit	A>kermit
MS-Kermit>set port 1	MS-Kermit>set port 1
MS-Kermit>set speed 9600	MS-Kermit>set speed 9600
MS-Kermit>set local on	MS-Kermit>set local on
MS-Kermit>set term newline on	MS-Kermit>set term newline on
MS-Kermit>connect	MS-Kermit>connect
Hi, can you read this?	Hi, can you read this?
Yes, I can read it!	Yes, I can read it!
Alt-X	*Alt-X*
MS-Kermit>exit	MS-Kermit>exit

If the Test Failed

- Is your cable securely attached at all points?
- Are you sure you have a null modem cable or a cable with a null modem adapter? If you have a null modem adapter, try removing it. If you don't have one, buy one.
- Are you sure the connector on the back of the PC that your cable is plugged into is a communication port and not a printer port?
- Are you sure your PC *has* a communication port? If it doesn't, you'll have to go to the computer store and get one.
- Are you sure the cable is connected to communication port number 1 (COM1) and not, for example, to COM2? Try repeating the example above, but use SET PORT 2 rather than SET PORT 1 (on one PC at a time).

If none of these questions produces an explanation of the problem, then several possibilities remain:

- Your serial port is broken. Read the technical manual for your communication device, and then run any available diagnosic tests. If it fails, it must be fixed or replaced.
- Your serial port is misconfigured. Read the installation manual that came with it. For example, you might have installed it as COM1 when there already was another COM1. If you have two ports, one of them should be configured as COM1 and the other as COM2.
- Your cable is defective. It must be fixed or replaced. Read the remainder of this chapter for some cable testing procedures.
- Some other software is interfering with Kermit. Look in your `CONFIG.SYS` and `AUTOEXEC.BAT` files for mouse drivers, alarm clock drivers, pop-up utilities, keyboard drivers, terminate-and-stay-resident (TSR) programs, and so on. Remove them all. Repeat the Kermit test. If it works, start putting back your other software, one program at a time, until Kermit stops working again. Then you know which program is the culprit. Don't run the offending program at the same time as Kermit.

Problems can occur in any combination, particularly in an old or inherited PC, and different (or the same!) problems can occur on both PCs at once. Finding and fixing one problem won't necessarily fix your communications. Be patient, keep trying.

Pinpointing the Problem

You can't communicate at all. Is the problem Kermit, the PC, the port, the connector, the cable, the other computer, or something else? There is an easy way to narrow this down: the *loopback connector*. A loopback connector plugs into your serial port. It has no cable. Inside the connector, the receive and transmit signals are connected. This means that whenever the PC sends a character, it immediately receives the same character. Loopback connectors are available from computer supply catalogs and computer stores. If you can find one, connect it to your port, and then run the following test:

```
A>kermit                          (Run MS-DOS Kermit)
MS-Kermit>set port 1              (Select the right port)
MS-Kermit>set local off           (Let loopback do the echoing)
MS-Kermit>connect                 (Begin terminal emulation)
abcdefghijklmnopqrstuvwxyz        (Type some characters)
Alt-X                             (Escape back to the PC)
MS-Kermit>exit                    (Exit to DOS)
A>
```

After giving the CONNECT command, type some characters, such as the alphabet shown in the example. If you see these characters on your screen, Kermit is working, your PC is working, and the port is working; therefore, the problem must be in the cable or the other computer. Move the loopback connector to the other PC and perform the same test. If it works, the problem must be in the cable or modem eliminator. If a PC fails the loopback test, the most likely causes are:

- Kermit is trying to use a communication port different from the one that the loopback connector is on. Use Kermit's SET PORT command to try different ports, or move the loopback connector. If you still can't make it work, then:
- There is no serial port. You have your loopback connector plugged into something else. Remove the loopback connector and go out and buy a serial port.
- The serial port is misconfigured. If you have one serial port, it should be configured as COM1. If you have two, they should be configured as COM1 and COM2. Read the installation instructions that came with your serial port.
- The serial port is broken. Get it fixed or buy a new one.
- The slot your serial port is plugged into is bad. Remove the cable from the port, turn off your PC, take the cover off your PC, pull out the serial port card, and plug it into a different slot. If you don't have a spare slot, plug it (firmly) back into the same slot. Run the test again. If you still can't communicate, your PC might need repair. Visit your dealer or contact a local computer club for more assistance.

To check the cable itself, plug one end into the PC and attach the loopback connector to the other end. Repeat the test. If it works, the problem is in the modem eliminator, and our loopback connector is a problem eliminator.

PC to Host Computer

If you have a direct cable connection (with a null modem) between your PC and a multi-user host computer (for example, a VAX/VMS system as in Figure 5-2), you can check the cable by running the Kermit program on the PC, setting the appropriate speed, and then attempting to log in, as in this example:

```
A>kermit                          (Run MS-DOS Kermit)
MS-Kermit>set speed 9600          (Set the speed)
MS-Kermit>connect                 (Begin terminal emulation)
Welcome to the VAX/VMS system     (Press Enter to get greeting)

Username: xxxxxx                  (Type some characters)
Alt-X                             (Hold down Alt and press X)
```

What happened?

- You saw the greeting message of the multiuser computer; the cable is fine, the port is fine, Kermit is fine. Proceed to Chapter 6.
- If, after typing the CONNECT command, you saw nothing, the trouble may lie in the cable, port, or PC, as described in the previous section of this chapter. Go back and read it. Try the loopback test if you can. It is also possible that the host port is broken or misconfigured. You will have to check with the administrator of the host computer.
- If you saw something, but it was not recognizable, Kermit's communication parameters might need to be adjusted to what the host expects. But you don't know how to do this yet. Skip the rest of this chapter and read Chapters 6 and 7. Then set up Kermit correctly for your host and try again.

Figure 5-2 Testing the PC-to-Host Connection

Figure 5-3 Testing a Modem Connection

Modem Connections

If you are using a modem, internal or external, your PC must make other connections before it can communicate with another computer. First, the PC must be able to talk to your modem, then the modem must be able to dial a telephone number and go across the telephone wires to talk to the other computer's modem, and finally the other modem must be able to talk to the other computer. (See Figure 5-3.)

This sounds more difficult than it is because normally you need to worry only about connecting your PC to your modem and instructing the modem to dial the number. Either the other modem will pick up and you will be connected or you will get an informative message like `BUSY` or `NO ANSWER`.

Modern modems include an autodialer that will dial the phone number for you. Though different modems work in different ways, the most popular method is the one used by Hayes Smartmodems and copied by the many other manufacturers that claim to make "Hayes-compatible" modems.

Hayes dialing commands are sent to the modem by the PC. (You do not press them on your telephone keypad!) The commands begin with the letters `AT`. If you type the com-

mand AT and then press the Enter key, the modem should respond with the message OK. If you do not see the OK message on your screen:

- Make sure your modem is turned on.
- Check that your cables are plugged in tightly.
- Try resetting the modem with the command ATZQ0V1 (see your Hayes manual).

The letters that follow the modem command AT tell the modem what to do; DT means Dial Touch-tone, and DP means Dial Pulse (rotary). After the modem dialing command ATDT or ATDP comes the telephone number. The modem command is not executed until you press the Enter key. For example:

ATDT7654321

This is the Hayes modem command for dialing the telephone number 765-4321 on a Touch-tone telephone.

In this book, dialing will always be illustrated by using Hayes commands since this method is so common. See Table I-2 for a listing of commonly used Hayes commands. If you have a different type of autodial modem, substitute your own modem's dialing commands. If your modem does not have a built-in dialer, you must dial the telephone number manually, just as you would for a normal telephone call. Consult your modem manual for details.

Dialing Up an Information Service

Preliminary details to consider:

1. Is your modem Hayes compatible?	A. ❒ Yes	B. ❒ No
2. What is your modem's speed?	C. ❒ 1200	D. ❒ 2400
3. How does your phone dial?	E. ❒ Touch-tone	F. ❒ Pulse (Rotary)

Now let's try to call up the Digital Equipment Corporation Electronic Store. You don't have to buy anything; you don't even have to tell them who you are—"Just looking!" Here's what you do:

1. Make sure your modem and your PC are turned on[7] and your modem is connected as described in Chapter 4 (did you read it?).

[7]Some external modems receive their electrical power from the communication line and don't need power from an electrical outlet. Such modems might not have an on/off switch, so you won't need to turn them on. See your modem manual.

2. At the DOS prompt, type `kermit`, and then press the Enter key:

```
C>                              (The DOS prompt)
C>kermit                        (Type KERMIT, press Enter key)
```

   If you see the message `Bad command or file name`, you didn't install Kermit correctly. Go back to Chapter 2, check your installation, and then come back here.

3. You will see MS-DOS Kermit's greeting and prompt:

```
IBM-PC MS-Kermit: 3.11
Copyright (C) Trustees of Columbia University 1982, 1991
Type ? or HELP for help
MS-Kermit>                      (This is Kermit's prompt)
```

4. At this point, you can type any Kermit command. Let's begin by setting the transmission speed:

```
MS-Kermit>set speed 2400
```

   If you checked box C above, type `set speed 1200` instead.

5. Now connect to the modem and type the dialing command. If you checked box B above, substitute your particular modem's dialing command or dial the number manually on your telephone. Otherwise, type only one of the ATD commands shown below, depending on the type of dialing your phone does. Be sure to dial the right phone number, or you might hear someone answer "Hello?" on your modem speaker.

```
MS-Kermit>connect               (Connect Kermit to your modem)
AT                              (Type AT, then press Enter)
OK                              (Modem should respond OK)
ATDT 1-800-234-1998             (Hayes with Touch-tone, Box E)
or
ATDP 1-800-234-1998             (Hayes with pulse dial, Box F)
```

   If you are dialing from a PBX, remember to include your PBX's prefix for an outside line. If the modem doesn't respond with `OK`, make sure it's turned on and properly connected (see "Connecting Your PC to an External Modem" in Chapter 4). If it is, go back and read the section "PC to PC" earlier in this chapter.

6. Wait about 30 seconds. If the call is answered successfully, you should see one of the following messages (if you have a Hayes or Hayes-compatible modem):

```
CONNECT
CONNECT 1200
CONNECT 2400
```

   If the message shows a different speed, like 1200 when you dialed at 2400, you must tell Kermit that the connection speed has changed behind its back, for example:

```
CONNECT 1200
Alt-X                           (Hold down Alt key and press X)
MS-Kermit>set speed 1200        (Change the speed to 1200)
MS-Kermit>connect               (Connect back to modem)
```

If the message was BUSY or NO ANSWER, wait a little while, and then go back to step 5 and try again.

7. Now press the Enter key and wait a few seconds. If you don't see anything, press the Enter key again. Repeat several times until you see a greeting, and then follow the directions.

Select the menu item "View a Demonstration of the Electronic Connection." You will see a nice demonstration of MS-DOS Kermit's DEC VT terminal emulator. Just follow the instructions on the screen. Note that <RETURN> means you should press the Enter key, and CTRL_C means you should hold down the Ctrl key and press the letter C.

You can wait for the demonstration to finish, or you can leave it prematurely. In either case, you should terminate your session by hanging up, just as you would on a regular phone call. This concludes our test drive. Exit Kermit and return to DOS as follows:

```
Alt-X                          (Hold down the Alt key and press X)
MS-Kermit>hangup               (Get Kermit prompt and type HANGUP)
MS-Kermit>exit                 (Exit from Kermit)
C>                             (Back at the DOS prompt)
```

Modem Problems

If you have a Hayes or other autodial modem, but you never saw anything on your screen after you gave the CONNECT command, you must face all the possible problems listed in the section "PC to PC" at the beginning of this chapter. Use the methods described there to pinpoint the problem.

In addition, your modem may be turned off, not working, or not configured correctly. Read your modem manual. Try your modem on somebody else's PC to see if it works there. If it doesn't, the modem needs reconfiguring, fixing, or replacing.

You can also use an external modem to test the serial port. Most external modems have status lights, which include a receive (RD or RxD) light and a transmit (SD, TD, or TxD) light. After giving the Kermit CONNECT command, type some characters. Do you see the transmit light blinking? If not, something is wrong with your serial port or the cable to your modem, or you have selected the wrong PC communication port.

Now let's find out more about MS-DOS Kermit's commands.

Chapter 6

Running MS-DOS Kermit

Now you have a working copy of the Kermit program for your PC and a tested data connection to another computer. So far, so good. Now we can discuss the MS-DOS Kermit software program itself.

There are two things you should find out whenever you plan to use a computer software program: how to start it and how to exit from it. You already saw how to do both of these in previous sections, but let's go over it one more time.

Starting and Stopping the Kermit Program

To run the MS-DOS Kermit program on your PC, type `kermit` at the DOS prompt, and then press the Enter key:

```
C>kermit
IBM PC MS-Kermit V3.11
Copyright (C) Trustees of Columbia University 1982,1991
Type ? or HELP for help

MS-Kermit>
```

At the `MS-Kermit>` prompt, you could type Kermit commands if you knew what they were. If an error message:

```
Bad command or file name
```

appears instead of the `MS-Kermit>` prompt, go back and read Chapter 2.

Some good commands to try are EXIT or QUIT, either of which will get you out of the MS-DOS Kermit program and back to the DOS prompt:

```
MS-Kermit>exit
or
MS-Kermit>quit
```

Practice going back and forth between DOS and Kermit a few times. Remember, the part you type is underlined, and you should terminate the underlined command by pressing the Enter key:

```
C>kermit
MS-Kermit>exit
C>kermit
MS-Kermit>quit
C>kermit
MS-Kermit>ex
C>kermit
MS-Kermit>q
C>
```

That wasn't too hard, was it? Notice that you can abbreviate EXIT as EX and QUIT as Q. In fact, you can abbreviate any Kermit command word (except a filename) as much as you like, as long as you have given enough letters to distinguish it from any other word that could appear in that field.

Menu on Demand

Unlike many other PC applications, Kermit does not present you with a full-screen menu, brightly colored windows, or sound effects when you run the program. It could, but it doesn't for reasons you may already understand if you make frequent use of menu-oriented software programs. Instead, Kermit gives you a prompt, `MS-Kermit>`, to indicate it is ready for a command. In response, you type a command. Kermit executes it and then gives you the next prompt. This goes on until you exit from the program.

Kermit's style is designed to be consistent across the hundreds of different kinds of computers and operating systems that Kermit programs run on. Once you have mastered one Kermit program, you have virtually mastered them all. And once you are familiar with the commands you need, you are not slowed down by screens full of menu options.

A typical Kermit command is an English verb, like HELP or EXIT. Some commands must be followed by additional words, known as *operands* or *parameters:* SHOW MODEM, CLOSE SESSION, and the like. Commands may even consist of several words: SET SPEED 2400, IF FAILURE GOTO JAIL. The command is terminated, and begins to execute, when you press the Enter key.

These commands resemble English sentences, and most of them make sense to the casual reader. As you might guess, there are hundreds of possible combinations of commands and operands. Without a menu, and without thumbing through a thick reference manual, how do you know how to type a command?

The key that unlocks the mystery of Kermit's commands is the question mark. At the `MS-Kermit>` prompt, you can type a question mark at almost any point, and Kermit will tell you what is possible or expected next. If you type a question mark in response to Kermit's prompt, Kermit lists all its major commands. If you type a major command followed by a question mark, Kermit tells you what sort of operand to type. Let's begin by seeing which SET command options begin with the letter S, and then let's pick one and find out what *its* options are, and so on, until we've completed a command:

```
MS-Kermit>set s? One of the following:
  send   server  speed
MS-Kermit>set server ?  One of the following:
  login  timeout
MS-Kermit>set server timeout ? Seconds, 0-255
MS-Kermit>set server timeout 0 ? Press ENTER to execute command
```

In this way, you can feel your way through a command by typing a question mark within each field. Notice that previously typed characters do not need to be retyped.

Editing Your Commands

You will see a dashed line or rectangular block blinking at you when you are using DOS or software programs like MS-DOS Kermit; this is called the *cursor*. As you type, the cursor moves along like the bouncing ball used on TV sing-along shows to help viewers follow along. The cursor positions itself where you stopped typing so you can always tell where you left off.

MS-DOS Kermit lets you edit your commands while you are typing them, as long as you haven't pressed the Enter key yet. Editing is done using the Backspace key or control characters. To type a control character, hold down the key marked Ctrl and press the indicated key. For example, to type a Ctrl-C character, hold down the Ctrl key and press the C key (uppercase or lowercase, it doesn't matter). Here are Kermit's command editing characters:

Backspace Deletes the *character* to the left of the cursor. May be typed repeatedly to delete characters all the way back to the `MS-Kermit>` prompt.

Ctrl-H Works the same as Backspace.

Ctrl-W Deletes the *word* to the left of the cursor. May be typed repeatedly to delete words all the way back to the `MS-Kermit>` prompt.

Ctrl-U Deletes the entire *line* all the way back to the `MS-Kermit>` prompt.

Ctrl-C Cancels the current command and returns immediately to the `MS-Kermit>` prompt.

Esc Completes the current word automatically if possible, and then goes on to the next word. If the rest of the current word cannot be guessed, the program beeps.

Summary of MS-DOS Kermit Command Features

- *Help is available for all fields*. You may type a question mark at any point within a command to get a short menu, except inside a filename (where a question mark acts as a wildcard character).
- *Kermit commands are case independent*. Commands and filenames may be in either uppercase or lowercase, or any mixture of them. Case does not matter.
- *Kermit commands may be abbreviated*. Any command word (but not a filename) may be abbreviated by omitting characters from the end as long as the abbreviation is different from any other word that can appear in the same field. For example, SER is sufficient to abbreviate SERVER, RU to abbreviate RUN, and SP to abbreviate SPEED.
- *Kermit command fields can complete themselves*. If you press the Esc key while typing a command word (but not a filename), and if the word is uniquely specified, Kermit will complete the word for you and position itself for the next word, if any. If the abbreviation is ambiguous, or if the word is a filename, Kermit will beep, and you may continue typing.
- *Kermit commands are not executed until you press the Enter key*. You always have the chance to ponder your command before activating it.
- *Kermit commands may be edited before entry*. While typing a command, you may use the Backspace key to erase characters back to the prompt. And you may press `Ctrl-W` to erase a word, `Ctrl-U` to erase an entire command, or `Ctrl-C` to cancel a command.

You should now take a few minutes to get comfortable with Kermit's command style, and the best way to do this is to get some practice. Following along on your PC, at the DOS prompt:

1. Type `kermit`. Did the Kermit program start? If not, did you remember to press the Enter key? If you did press the Enter key, you probably have not installed the program correctly. Go back to Chapter 2, and pay attention this time!
2. At the `MS-Kermit>` prompt, type `?` (question mark). You will see a list of all the MS-DOS Kermit commands, with a brief description of each. You will probably

never need most of these commands, but each has its purpose. (See Chapter 17 for a complete description of all Kermit's commands.)

3. At the MS-Kermit> prompt, type set, followed by ? (question mark). These commands give you ways to change Kermit's behavior. Don't be frightened; usually Kermit behaves correctly without any extra instruction. Now press *Ctrl-U* to erase the SET command.

4. Type the command set terminal vt102 one letter at a time, following each letter with ? (question mark), and watch how the choices narrow down.

5. Now type the same command again, but this time press the Esc key after each letter instead of a question mark.

6. Type exit, and then press Enter to quit Kermit and return to the DOS prompt.

Kermit Startup Options

As you have seen, Kermit is normally an *interactive* program. You talk to it, and it talks to you. You can also make it act like a typical DOS "one liner" command if you include Kermit commands on the DOS command line that invokes Kermit, for example:

```
C>kermit set speed 19200, send oofa.c
```

Multiple commands are separated by commas. When invoked in this manner, Kermit executes all the commands on the command line and then exits to DOS without ever showing its prompt. If you want to give command line options, but still get the prompt, include the STAY command:

```
C>kermit set speed 19200, send oofa.c, stay
```

You can use this feature to write DOS batch commands that start Kermit in special ways. For example, the file C.BAT might contain the line:

```
kermit set speed 9600, connect, stay
```

If C.BAT is stored somewhere in your PATH, just typing the C command at the DOS prompt will start Kermit and connect you to the host.

On the MS-DOS Kermit diskette, you will find a file called MSKERMIT.INI. This is the program's initialization file and includes Kermit commands. Each time you start up Kermit, these commands are executed automatically so you don't have to type them at the MS-Kermit> prompt. This file must be stored in your PATH or in your current directory. The most natural place to keep the initialization file is in the same directory with the KERMIT.EXE file. The initialization file can be altered (or created if one doesn't exist already) with a text editor after you become more familiar with Kermit's commands.

Some Basic Kermit Commands

You can experiment with some Kermit commands right away since they don't involve data communication at all. MS-DOS Kermit has several important DOS commands built in. These include CD (change directory), DIR (which you can spell out as DIRECTORY if you like), TYPE, and DELETE.

```
MS-Kermit>cd files
MS-Kermit>dir oofa

 Directory of  C:\FILES

OOFA     TXT      9753   9-21-89   8:43p
OOFA     OBJ       864  10-17-89   4:24p
OOFA     EXE      2861  10-17-89   4:24p
        3 File(s)    720 bytes free

MS-Kermit>type oofa.txt
  (The file OOFA.TXT is displayed)

MS-Kermit>del oofa.obj
MS-Kermit>cd \kermit
```

And the RUN command lets you run absolutely any DOS command or program from within Kermit, provided your PC has sufficient memory. Just type the desired DOS command after the RUN command, and then press the Enter key:

```
MS-Kermit>run basic
MS-Kermit>run rename oofa.txt secret.txt
MS-Kermit>run dir | find "<DIR>" | sort
```

Before proceeding to Chapter 7, practice using the features of Kermit's command processor. Get used to using a question mark, abbreviations, *Ctrl-U*, *Ctrl-W*, *Ctrl-C*, Esc, Backspace, and Enter so the examples to come will be easy for you.

Chapter 7

Getting Online

We've had a sampling of MS-DOS Kermit commands but have not learned which ones are absolutely needed and which are there to make life easier.

Like our ecclesiastical friends, the priest, the rabbi, and the minister, computers of different persuasions must adapt to one another's quirks. Fortunately, a PC equipped with Kermit is flexible and open-minded—ecumenical to a fault. Whenever it must communicate with a less tolerant computer of a different faith, MS-DOS Kermit will happily "convert" given the proper instruction.

The magic word is SET. SET this, SET that, then CONNECT, and poof, you're online! Please pardon the lack of detailed explanation of the following tenets of data communications. These are mysteries you need not bother with to use MS-DOS Kermit.

Setting Communication Parameters

As you might expect, the Kermit command SET defines or changes the characteristic you specify. The characteristic it defines or changes is called a *parameter*. There are several parameters you can set. You already know about the parameter called SPEED. In Chapter 5, you learned that the MS-DOS Kermit program and your modem must use the same speed in order to communicate and how to instruct MS-DOS Kermit to use the correct speed:

```
MS-Kermit>set speed 1200
```

Kermit's SET parameters have names like SPEED and BELL. Each of these parameters can have options too. The parameter options are referred to as *values*. In the example above, the parameter that is SET is SPEED, and the value it is set to is 1200.

Certain parameters are essential to establishing a successful connection if both terminal emulation and file transfer are to work correctly. Kermit parameters are set automatically, without any interaction from you, when the MS-DOS Kermit program is started. These startup values are called the *default* parameters. A Kermit SET command can override these default parameters either directly at the `MS-Kermit>` prompt or when included in your `MSKERMIT.INI` initialization file.

The most important parameter of all is the communication port. Unless you tell Kermit otherwise, it will use COM1, communication port number 1. COM1 can be either a serial port or an internal modem. If you need to use some other kind of communication device or communicate over a local area network, other options are available (see page 232). Let it never be said that Kermit limits your options.

Accessing the Port

An IBM PC or PS/2 can have more than one communication port. These ports are known by the DOS device names COM1, COM2, COM3, and COM4 and may be any combination of serial ports and internal modems.

Most serial ports and internal modems made for IBM PCs, PC/XTs, and PC/ATs can be configured as any of these COM devices. This is done on the card itself by setting tiny switches or by plugging or unplugging a small jumper block. If you have only one communication port, you should make sure it is configured as COM1. If you have two, you should make sure that one of them is set up as COM1 and the other as COM2. Consult your hardware reference manual for further information.

Serial ports and internal modems for the PS/2 are self-configuring and, presumably, cannot be installed incorrectly. The first serial port, COM1, comes as standard equipment on every PS/2. Additional serial ports, such as the IBM Dual Async Adapter A, should be configured using the PS/2 Reference Diskette. See your PS/2 Quick Reference manual for instructions.

For MS-DOS Kermit (or any other software) to communicate successfully, it must know which port to use and how to use it. More often than not, COM1, which Kermit tries to use unless you say otherwise, is the correct choice. If you did not install the board yourself, the only way to know for sure if your serial port is COM1, COM2, COM3, or COM4 is by trial and error. This chapter discusses only the PC's serial ports. See page 232 for other communication options, and see Chapter 16 for how to establish local area network connections.

The command for selecting a communication device is SET PORT:

SET PORT COM1

(or SET PORT 1) Select communication port 1. You may also specify COM2, COM3, or COM4 (or 2, 3, or 4) if you are using a different port. If the port is an IBM serial port (an 8250 or 16550A UART), or another device (such as an internal modem) that perfectly mimics an IBM serial port, Kermit controls it directly for high-speed, efficient operation. Otherwise, Kermit uses the services of the PC's Basic Input Output System (BIOS), which results in slower operation and fewer features (for example, the inability to send a BREAK signal). The default port is COM1, meaning COM1 is used if you don't issue a SET PORT command at all.

If your PC has more than one port, you must be sure that Kermit is trying to use the correct one. If Kermit is using COM1 but your cable is connected to COM2, there will be no communication. You must tell Kermit to use COM2 instead, for example:

```
MS-Kermit>set port com2
or
MS-Kermit>set port 2
```

If your PC has a nonstandard port configuration, Kermit might not be able to find or use the port you have specified. This is especially likely if the port is COM3 or COM4. If this happens to you, read about nonstandard communication ports on page 207 and the SET COM*n* command on page 228 in the command summary.

The SET PORT command has a special property. The communication settings listed below—speed, duplex, flow control, handshake, and parity—are remembered for each port. The related SET commands affect the current port only. If you issue a SET PORT command to change ports, you get the new port and all its communication parameters. You can switch among ports as often as you like using only the SET PORT command.

Communication Parameters

Once you have chosen the communication port, you must be sure that the communication parameters that affect it are set appropriately:

SET SPEED

Two data devices cannot communicate *at all* unless they use the same transmission speed. The common speeds are 300, 1200, or 2400 bits per second (bps), or baud, for dialup (modem) connections; 4800, 9600, or 19200 bps for local PC-to-host connections; and 38400 or 57600 bps for short, direct PC-to-PC connections. Under exceptional conditions, two PCs might be able to communicate through a very short shielded cable at 115200 bps. MS-DOS Kermit's speeds can be abbreviated, just like words. For example, since 9600 is the only speed that begins with 9, SET SP 9 is equivalent to SET SPEED 9600.

```
MS-Kermit>set speed ? One of the following:
  45.5  50  75  110  134.5  150  300  600  1200  1800
  2000  2400  4800  9600  19200  38400  57600 115200 75/1200
MS-Kermit>set speed 9
```

The transmission speed applies only to COM ports, not to BIOS or network (except EBIOS) connections (see Chapter 16). MS-DOS Kermit does not have a built-in default for transmission speed. It uses whatever speed the port was left at the last time it was used unless it is told otherwise.

SET DUPLEX

Data connections can be either *full duplex* or *half duplex*. Full duplex means that both computers are allowed to transmit at the same time, and each computer is capable of simultaneously sending and receiving data, much like a conversation between two New Yorkers. This style of connection is the most common. It is used by UNIX systems, DEC VAX computers with VMS, and many others. On a full-duplex terminal connection, the characters you type on your keyboard are sent to the other computer, and the other computer "echoes" them back to you. This is called *remote echo*.

On a half-duplex connection, often used with IBM mainframes, only one computer is allowed to transmit at a time, much like CB radio. The two computers must take turns, and the characters you type are echoed to your screen by your own terminal (or in our case, by MS-DOS Kermit). This is called *local echo*.

The MS-DOS Kermit command is SET DUPLEX, with the options FULL (for full duplex) and HALF (for half duplex).

```
MS-Kermit>set duplex ? One of the following:
  full  half
MS-Kermit>set duplex half
```

SET DUPLEX HALF automatically enables RTS / CTS (Request To Send / Clear To Send) hardware line-turnaround for use with half-duplex modems, spread spectrum radios, and other devices that enforce two-way-alternate communication by using the RTS and CTS modem signals.

SET LOCAL-ECHO

This command turns local echoing ON and OFF. Unlike SET DUPLEX HALF, the SET LOCAL-ECHO ON command does not also enable the use of RTS / CTS to enforce half duplex operation.

SET FLOW-CONTROL

Back in Chapter 3, did you try using *Ctrl-S* (hold down the Ctrl key while pressing the letter S) and *Ctrl-Q* to stop and resume the display of a long file during the DOS TYPE command? If not, try them now. Two computers with a full-duplex connection can be configured to do this to each other automatically. The process is called *Xon/Xoff flow control*. If computer A is receiving data faster than it can handle, it

sends an Xoff (Ctrl-S) to computer B, and computer B stops transmitting. When computer A has caught up, it sends an Xon (Ctrl-Q), and computer B resumes transmitting. The Kermit command for this is SET FLOW XON/XOFF. You should use Xon/Xoff flow control if the other computer supports it. To test this, connect to the other computer, TYPE a file, and then try typing *Ctrl-S* and *Ctrl-Q* manually to see if these characters suspend and continue the display.

```
MS-Kermit>set flow ? One of the following:
  None, Xon/Xoff, RTS/CTS
MS-Kermit>set flow xon/xoff
```

RTS/CTS (Request To Send / Clear To Send), sometimes called "hardware flow control," is another kind of flow control. It is usually used with high-speed modems, terminal servers, or digital PBXs. Give the command SET FLOW RTS/CTS if *(a)* Xon/Xoff does not seem to prevent data loss; *(b)* you have a full-duplex connection; *(c)* the device to which your PC is directly connected also supports RTS/CTS flow control; and *(d)* that device is configured to use it.

Xon/Xoff is normally used *end-to-end* between your PC and the other computer. RTS/CTS is between your PC and the device it is directly connected to. RTS/CTS and Xon/Xoff cannot be used simultaneously.

SET HANDSHAKE

On a half-duplex connection, where only one computer is allowed to transmit at a time, there must be a signal by which each computer turns over transmit permission to the other. This is called a *line turnaround handshake*. The terminal or PC grants permission by sending a carriage return (for example, when you press the Enter key). The computer on the other end uses another special character (usually not a carriage return), such as Xon (Ctrl-Q). The Kermit command is SET HANDSHAKE. The operands are NONE, XON, and several others.

```
MS-Kermit>set handsh ? One of the following:
  xon, none ... (several others)
MS-Kermit>set handshake none
```

The handshake setting is ignored during terminal emulation, but it is important during file transfer.

SET PARITY

Our final mysterious parameter is *parity*. Computer data is usually stored in chunks of eight bits. A bit is the basic unit of information in a computer and can have only two values: 0 and 1. An eight-bit chunk is called a character, or byte. The byte is the basic unit of transmission. But here's the tricky part. Although many computers transmit all eight bits of each byte, others transmit only seven and use the eighth bit for a kind of error check called parity. You must match MS-DOS Kermit's parity with that used by the other computer. The command is SET PARITY, and the operands are NONE, EVEN, ODD, MARK, and SPACE.

```
MS-Kermit>set parity ? One of the following:
  none, even, odd, mark, space
MS-Kermit>set parity none
```

Start Bits, Stop Bits, Word Length, Data Bits

You will often see these terms in literature that describes how to set up your communication software for particular services. Kermit always uses the appropriate number of start and stop bits, based on the connection speed; there is no SET command to change the number of start and stop bits. Kermit's word length (or data bits) is 8 if PARITY is set to NONE, and 7 if PARITY is set to anything else. Kermit cannot be configured to use 8 data bits plus parity.

The DUPLEX, LOCAL-ECHO, FLOW-CONTROL, and HANDSHAKE combinations are important but also a bit confusing. For full-duplex, remote-echo connections (the most common type), all the default settings apply: SET DUPLEX FULL, SET LOCAL-ECHO OFF, SET FLOW-CONTROL XON/XOFF, SET HANDSHAKE NONE. If you are using a high-speed modem or other device that supports RTS/CTS flow control on such a connection, you can change the FLOW-CONTROL setting to RTS/CTS.

For half-duplex connections (such as IBM mainframe linemode sessions), you would SET LOCAL-ECHO ON, SET FLOW-CONTROL NONE, and SET HANDSHAKE XON. If, in addition, your PC is directly connected to a half-duplex device (a half-duplex modem or radio) that controls line access with the RTS and CTS signals, you should also SET DUPLEX HALF.

Remember, the SET commands for speed, duplex, flow control, handshake, and parity apply to the currently selected serial port—the port you specified in your most recent SET PORT command, or else to COM1 if you have not given a SET PORT command.

Before you can communicate successfully with another computer, you must find out its communication parameters and set them appropriately in MS-DOS Kermit. Table 7-1 should help. (A few entries are left blank in case you want to fill them in with your own favorite computer hosts.) Speed is not shown in the table because it depends on your connection method—1200 bps modem, 2400 bps modem, direct connection, and so on.

Sometimes you will have to set a lot of parameters yourself, and sometimes the default settings will apply. It depends on the computer you want to communicate with and your method of connection. To set up MS-DOS Kermit for a direct 9600 bps connection to a DEC VAX/VMS system, the only parameter you must specifically SET is SPEED:

```
MS-Kermit>set speed 9600        (All defaults apply)
```

At the other extreme, to prepare for a 2400 bps dialup linemode connection to an IBM mainframe, you must SET many parameters:

```
MS-Kermit>set speed 2400          (Set the speed)
MS-Kermit>set duplex half         (Half duplex)
MS-Kermit>set flow none           (No flow control)
MS-Kermit>set handshake xon       (Xon handshake)
MS-Kermit>set parity mark         (Mark parity)
```

The communication settings used in Table 7-1 and the examples above are typical but may differ from those used at your site. Check the documentation for the computer you will actually be connecting to. This information should be readily available from the people who maintain that computer.

Dialup services like CompuServe and Dow Jones generally use the parameters listed above under Direct Dial Services, but the parity setting might be something other than NONE. Check your subscriber literature. Since there are only five possible parity settings, you can try each until you find the one that works.

Table 7-1 Typical Communication Parameters

Environment	*Duplex*	*Flow*	*Handshake*	*Parity*
Default	Full	Xon/Xoff	None	None
UNIX	Full	Xon/Xoff	None	None
VAX/VMS	Full	Xon/Xoff	None	None
PDP-11	Full	Xon/Xoff	None	None
IBM Mainframe Fullscreen	Full	Xon/Xoff	None	Even
IBM Mainframe Linemode	Half	None	Xon	Mark
MS-Kermit to MS-Kermit	Half	Xon/Xoff	None	None
Direct Dial Services	Full	Xon/Xoff	None	None
X.25 Public Network	Full	Xon/Xoff	None	Mark
Local Area Network	Full	Xon/Xoff	None	None
High-Speed Modem	Full	RTS/CTS	None	???
?				
?				
?				

Note the entry for X.25 Public Network. These are services like SprintNet (formerly Telenet) and Tymnet in the United States and Datapac in Canada. These networks generally provide only seven-bit transmission and mask the other characteristics of the remote computer from you. Again, the parameter settings might be somewhat different, so check your subscriber literature.

When MS-DOS Kermit is operating through a local area network (LAN), whether it is DECnet, IBM Token Ring, or a vendor-supplied TCP/IP interface, the connection will almost always use the default parameters (see Chapter 16 for details).

What Happens If the Parameters Are Wrong?

You must adjust the communication parameters on your PC to match those of the computer or service you are connecting to. Although your parameters may seem correct during terminal emulation, only successful file transfer will tell for sure. There are some parameters that immediately identify themselves when you try to use terminal emulation.

COMMUNICATION PORT

If you are using the wrong communication port, you will not be able to communicate with your modem or the remote computer at all. You might see a message like "Serial port COM3 not available". Or you may see only a blank screen after you type the Kermit CONNECT command, no matter what keys you press. Try a different port, for example, SET PORT 2 instead of SET PORT 1, or see pages 207 and 228 about nonstandard communication ports.

TRANSMISSION SPEED

If your transmission speed (baud rate) is set wrong, the speeds of the two computers will be mismatched, and you will see incomprehensible garbage on your screen, something like:

```
zj:Vo{{:n{::j{{:zbCzRrzrrz~{:~{{{{zqr::zj
```

Change your speed using SET SPEED. Try different speeds until you see comprehensible messages.

DUPLEX or **LOCAL-ECHO**

If you have the wrong duplex or local-echo setting, echoing will not be normal. If MS-DOS Kermit is set to full duplex (local echo off) when the connection is really half duplex, the characters you type will not appear on your screen at all, even though the responses to your commands will appear normally. Solution: SET DUPLEX HALF or SET LOCAL-ECHO ON. If the connection is full duplex and MS-DOS Kermit is set to half, every character you type will appear twice, for example:

```
wwhhaatt''ss ggooiinngg oonn hheerree??
```

Solution: SET DUPLEX FULL or SET LOCAL-ECHO OFF.

FLOW CONTROL

On a full-duplex connection, loss of data can be prevented by using Xon/Xoff flow control, but only if both computers are set up to do it. MS-DOS Kermit does this by default, and so do most computers, including UNIX-based computers, VAX/VMS, and others. If MS-DOS Kermit is doing Xon/Xoff but the other computer is not, an Xoff signal (Ctrl-S) sent by Kermit to the other computer will be misinterpreted, with unpredictable consequences. Solution: Tell MS-DOS Kermit to SET FLOW NONE.

If MS-DOS Kermit is not doing flow control, there is a danger of losing data during long, scrolling screen displays at high speeds and when Kermit is printing screens or saving them to a file. Flow-control signals sent by the other computer under these conditions will be accepted by MS-DOS Kermit as ordinary data characters. Solution: Tell MS-DOS Kermit to SET FLOW XON/XOFF.

If your PC is directly connected to a high-speed modem, terminal server, data PBX, or other device capable of RTS/CTS (hardware) flow control, be sure to enable this feature on that device, and give Kermit the SET FLOW RTS/CTS command. This prevents data loss even more effectively than XON/XOFF does. The effect of RTS/CTS is immediate and the signals are not subject to noise corruption.

On a half-duplex connection, be sure to SET FLOW-CONTROL NONE.

PARITY

Normally, MS-DOS Kermit tries to ignore parity as much as possible. During terminal emulation, MS-DOS Kermit strips the parity bit from each arriving character so even when its parity is set to NONE and the other computer is sending, say, MARK parity, the display will appear correct. The problem is usually in the other direction. If the computer on the other end, or some piece of equipment between the two computers, *needs* a certain kind of parity, and if MS-DOS Kermit doesn't send it, certain characters will not arrive correctly at their destination. We cannot always safely ignore parity during terminal emulation, and we *must* get it right for file transfer.

Dialing Your Modem

The usual method for dialing is to type your modem's dialing commands directly at the modem, such as ATDT5551212 for a Hayes modem, as illustrated in Chapter 5.

However, MS-DOS Kermit does include a script programming language to let you automate routine procedures such as dialing. Chapter 14 shows how to construct a DIAL command for Hayes modems, and the result is included in the `MSKERMIT.INI` and `HAYES.SCR` files that came with MS-DOS Kermit. If you've installed Kermit correctly (see Chapter 2), you can use them without further ado.

Here is Kermit's DIAL command for Hayes modems. If you have a modem that is not Hayes-compatible, you will have to read Chapter 14 to learn how to modify the `MSKERMIT.INI` and `HAYES.SCR` files to work with it.

DIAL *number*

If you have installed MS-DOS Kermit correctly, this command dials the specified telephone number using Hayes Smartmodem 2400 dialing commands. If you have dialed a number previously and you omit the telephone number from the DIAL command, the same number is redialed. If the line is busy, the call is placed again automatically after one minute; the process repeats up to five times until the call is answered. You can optionally prefix the phone number by T to force Tone dialing, or by P to force Pulse dialing.

Before dialing a call, be sure to set your communication speed to one that is supported by your modem, as well as any other necessary communication parameters. Here is an example:

```
MS-Kermit>set parity even       Parity needed by host
MS-Kermit>set speed 1200        Dialing speed for modem
MS-Kermit>dial 7654321          Dial the number
Dailing 7654321, wait...        Message from Kermit
CONNECT 1200                    Message from modem
MS-Kermit>                      Kermit's prompt returns
```

When you are finished with your call, give the HANGUP command to ensure that the telephone call hangs up:

```
MS-Kermit>hangup                Hang up the phone
MS-Kermit>exit                  Return to DOS
C>
```

High speed modems are becoming more affordable and popular. These modems support dialing speeds of 9600 bps, 19200 bps, or higher. MS-DOS Kermit can handle these speeds quite well with proper flow control, even on slow PCs. If your high-speed modem supports RTS/CTS hardware flow control, you should configure your modem to use it (see your modem manual) and tell MS-DOS Kermit to SET FLOW RTS/CTS:

```
MS-Kermit>set parity mark       Parity needed by host
MS-Kermit>set speed 19200       Dialing speed for modem
MS-Kermit>set flow rts/cts      Hardware flow control
MS-Kermit>dial T7654321         Tone-Dial the number
Dailing T7654321, wait...       Message from Kermit
CONNECT 19200                   Message from modem
MS-Kermit>                      Kermit's prompt returns
```

For further information about dialing, including instructions for setting up a dialing directory, read Chapter 14.

Using MNP Modems

Modern modems often include MNP (Microcom Networking Protocol) error correction. This can result in a clean connection even over noisy telephone lines if both modems use MNP. But if your modem uses MNP and the other modem does not, problems can occur when your call is answered. The most common symptom is that the remote host or service guesses your speed incorrectly, and then you can't communicate. This happens because your MNP modem sends a message to see if the other modem has MNP too. If the other modem does not have MNP, the message passes through to the host and can interfere with its normal speed recognition procedure. If this happens to you, follow the directions in your modem manual to disable the MNP feature before placing the call.

Even when MNP error correction has been successfully negotiated between the two modems, you still have no guarantee of an error-free connection. MNP error correction operates only between the modems, and not between the computers. If you see "garbage" on your screen during terminal emulation, check that your modem cable has not wiggled loose. During file transfer, Kermit's error checking takes over.

Some MNP modems are also capable of data compression. This does not necessarily result, however, in a speed improvement. If your MNP modem successfully negotiates data compression with another MNP modem, you will see increased performance only if the transmission speed between the modem and your PC is significantly higher than the transmission speed that is used between the two modems. For example, suppose you have an MNP modem capable of high-speed V.32 operation. If you configure your modem to "lock interface speed," and you SET SPEED 19200 in Kermit, and the other modem doesn't understand V.32 and therefore answers at a lower speed such as 2400, then MNP's compression will make it seem like you have a much faster connection than you really have.

All of this happens outside of Kermit. Kermit knows nothing about MNP or V.32, and has no built-in commands to control how your modem works. Consult your modem manual to learn how to set your modem up in the best way for any given type of connection, as well as the documentation for the remote computer or service you are dialing.

Chapter 8

Terminal Emulation

The world is bursting with computers and data services that possess a wealth of information and applications just for you. Most of these computers are designed to be accessed by terminals. But you have a PC. Luckily for you, your PC—when equipped with MS-DOS Kermit—can do everything a terminal can do, and much more.

Kermit's job is to create the best possible connection between your PC and the host computer. But because there are so many different kinds of computers, so many different kinds of terminals, and so many different ways in which terminals and computers communicate, this job is not always simple.

The Mechanics of Terminal Emulation

Before you can use Kermit as a terminal, you must set all of your communication parameters appropriately, as explained in Chapter 7. Once this is done, you can issue Kermit's CONNECT command. CONNECT is such a frequently used command that it has a special one-letter abbreviation, C.

Once you give the CONNECT command at the `MS-Kermit>` prompt, you have begun terminal emulation; your PC has become a terminal to the other computer. Terminal emulation gives you the impression that you are typing directly to the other, remote computer instead of to the PC on your desk. Any commands you type are ignored by your PC and are interpreted by the remote computer. For example, if you type the command DIR at the `MS-Kermit>` prompt, you will see a list of the files on the PC (in your current directory).

Remember that none of the commands you type is recognized until you press the Enter key.

```
C>kermit                          (Start up Kermit on your PC)
MS-Kermit>cd \kermit              (Change directory to \KERMIT)
MS-Kermit>dir                     (Request a list of files)

 Volume in drive C is EASYDISK
 Directory of  C:\KERMIT

KERMIT   EXE   112416   9-10-89   9:47p
MSKERMIT INI     3265   8-08-89   8:31a
KERMIT   PIF      165  12-09-89   5:36p
        3 File(s)   1472512 bytes free
```

After you have typed the CONNECT command at the `MS-Kermit>` prompt and have given the proper codes to gain access to the remote computer, a DIR command typed there will give completely different results.

Gaining access to a computer with more than one user is called *logging in*, and ending your access to this computer is called *logging out*. The access codes generally consist of a *username* and a *password*, which are issued to you by those who manage the multiuser computer.

```
C>kermit                          (Start up Kermit on your PC)
MS-Kermit>set sp 9600             (Set the right speed)
MS-Kermit>set parity even         (Maybe other parameters too)
MS-Kermit>connect
Welcome to the Other Computer, VMS V5.0-2
Username: kim                     (Type your user ID)
Password: ________                (Type your password)

$ dir                             (Type the DIR command)

Directory $DISK1:[KIM]

CALIF.DIR;1  CKERMIT.INI;9 EDTINI.EDT;6    EMACSINI.EL;1
LOGIN.COM;6  MAIL.MAI;1    MOON.DOC;5      OOFA.C;44
OOFA.OBJ;35  OOFA.EXE;35   VMSKERMIT.INI;8

$ logout                          (Remember to log out!)
```

These are entirely different files on a completely different computer.

When you CONNECT to the other computer, you are actually using your PC keyboard to talk to *two* computers—your PC and a remote computer. You need some way to shift the conversation from one computer to the other. The standard method is to type a special *escape sequence*, `Ctrl-]C`. That is, hold down the Ctrl key, press] (the right bracket

key), let go of the Ctrl and] keys, and then press the letter C (for Close Connection). Practice this a few times:

```
MS-Kermit>set speed 9600          (Set the right speed)
MS-Kermit>connect                 (Begin terminal emulation)

$ Now I'm communicating with the host computer.
Command not found: "Now"
Ctrl-]c                           (Escape back to Kermit on the PC)
MS-Kermit>connect                 (Connect to the host again)
$ help
UNIX helps those who help themselves.
$ thanks a lot!
Command not found: "thanks"
Ctrl-]c                           (Escape back to Kermit)
MS-Kermit>
```

When you reconnect, your previous terminal screen reappears. Escaping back (returning) to the PC does not terminate your session on the other computer. You must use whatever commands are necessary to end your session on the other computer. (A typical command for this purpose is LOGOUT.) If you are paying per minute for the connection, this is an important fact to remember. Even if time on the remote computer doesn't cost you a cent, keep in mind that turning off your PC for the night does not necessarily mean the connection to the other computer is gone. It might be possible for someone to start up your PC, run a copy of the MS-DOS Kermit program, type CONNECT, and access your host session. This is not a good thought if your manager is that "anyone" and your résumé is the file you were working on!

The Mode Line

Also notice the all-important *mode line* at the bottom of your screen, which looks approximately like this:

```
Esc-chr:^] help:^]? port:1 speed:9600 parity:none echo:rem VT320
```

This gives you the vital statistics on your terminal session. It tells you that your escape character is *Ctrl-]*,[8] you can get help about connect-mode escape sequences by typing *Ctrl-]* followed by a question mark, your communication port is COM1, your transmis-

[8]Control characters are often written as ^ followed by a letter or symbol, for example, ^A for Ctrl-A. They are often displayed in this way by host computers and even by DOS. To see this, type *Ctrl-A* at the DOS prompt. Note: on German keyboards, Ctrl-] is entered by typing Ctrl-+.

sion speed is 9600 bps, parity is none, character echo is being done remotely (full duplex), and Kermit is emulating a VT320 terminal.

Connect-Mode Escape Options

The escape character, *Ctrl-]*, can take other operands besides C. You can get a list of them by typing *Ctrl-]?*, just as it told you in the mode line above. For example, *Ctrl-]P* will "push" to DOS. This gives you a DOS prompt, and you can carry on with DOS as long as you like without forgetting your parameter settings for the host or terminating the connection. When you want to return to Kermit, type the DOS command EXIT , and you are back in your terminal session, exactly where you left off, with your previous screen restored.

Another useful connect-mode command is *Ctrl-]F*, which copies ("dumps") the current screen into a file on your DOS disk. The name of this file on your DOS disk is KERMIT.SCN, but you can select a different file with Kermit's SET DUMP command. To do this, return to the MS-Kermit> prompt by typing *Ctrl-]C* and give the command SET DUMP and a filename. Then issue the CONNECT command again, and type *Ctrl-]F:*

```
$
The screen looks like this on the other computer.
It is filled with lines of text.
And commands.
$
Ctrl-]c                          (Escape back to Kermit on the PC)

MS-Kermit>set dump amm.scn       (Select a new screen copy file)
MS-Kermit>connect                (Connect to other computer again)

Ctrl-]f                          (Ctrl-right bracket followed by F)
                                 (to copy screen to PC file AMM.SCN)
```

If the screen-copy file already exists, new material is added to the end, with each screenful separated by a Ctrl-L (formfeed) character. Otherwise, a new file is created the first time you copy the screen. To find out how you can save more than one screenful of data, see the sections on Screen Rollback and Session Logging later in this chapter.

Table 8-1 lists all the MS-DOS Kermit connect-mode escapes. The BREAK signal, which Kermit sends if you type *Ctrl-]B* (hold down the Ctrl key and press right bracket, then press the letter B), is not a normal character but a special signal lasting about one-quarter of a second that is used for various purposes by some hosts or communication processors. A long BREAK is the same thing, but it is longer and usually used for a different purpose, like breaking a connection.

Table 8-1 MS-DOS Kermit Connect-Mode Escapes

Escape	*Function*
`Ctrl-]?`	Display a list of escape functions
`Ctrl-]0`	(the digit zero) Transmit a NUL character
`Ctrl-]B`	Transmit a BREAK signal
`Ctrl-]L`	Transmit a long BREAK
`Ctrl-]C`	Return to the MS-Kermit prompt
`Ctrl-]H`	Hang up the phone; terminate a modem connection
`Ctrl-]F`	Copy the current screen into a DOS file
`Ctrl-]M`	Turn the mode line off if it is on; turn it on if it is off
`Ctrl-]P`	Push to DOS (use DOS EXIT to return to Kermit's terminal screen)
`Ctrl-]Q`	Temporarily quit logging the terminal session
`Ctrl-]R`	Resume logging the terminal session
`Ctrl-]S`	Show the status of the connection
`Ctrl-]Ctrl-]`	(two copies of the escape character) Send the escape character itself

Kermit's connect-mode escapes are hard to type. This is done on purpose to prevent you from accidentally invoking a function like hangup. For convenience, some of them also have Alt-key equivalents, and certain other functions are also assigned to Alt keys. Table 8-2 shows some of the Alt-key functions. To invoke an Alt-key function, hold down the Alt key and press the indicated key. For example, to return from terminal emulation to the `MS-Kermit>` prompt, hold down Alt and press X.

Table 8-2 MS-DOS Kermit Alt-Key Commands

Alt Key	*Function*
`Alt-X`	Exit from connect mode back to the MS-Kermit prompt
`Alt--`	(Alt-minus) Change Kermit's terminal type
`Alt-=`	Reset the terminal and clear the screen
`Alt-B`	Transmit a BREAK signal

Do I Need a Terminal Emulator?

Kermit's well-deserved reputation as a reliable and efficient file transfer protocol may mislead you into thinking that you need another package for high-quality terminal emulation. You don't.

If your only interest is transferring files between your PC and the remote computer, you probably know all you need to know about terminal emulation—just enough to get you connected. If so, go on to Chapter 9 and read about file transfer. But if you are connecting to a bulletin board or dialup service, or if you plan to actually use the remote computer to do work, you should become familiar with Kermit's terminal emulation features.

Terminal Types

Different terminals have different characteristics. The appearance of their screens is controlled in different ways by the computers they communicate with. For your screen to be formatted correctly, Kermit must emulate the kind of terminal that the remote computer believes it is controlling. MS-DOS Kermit can emulate any of the following terminals:

DEC VT102
The industry standard terminal. Features include direct cursor positioning, the ability to partition the screen into separate scrolling regions, character highlighting (boldface, underscore, inverse video), and editing capabilities such as line and screen erasure and line and character insertion and deletion.

DEC VT100
Like the VT102, but without the editing capabilities. MS-DOS Kermit emulates a VT102, but identifies itself to the host as a VT100.

DEC VT320
A more advanced DEC terminal that does everything the VT102 does but also includes the ability to switch among different character sets, enter international characters from the keyboard, and program its keys under host control. The VT320 has a PC-like keyboard that includes editing and function keys. Unless you say otherwise, MS-DOS Kermit emulates the VT320 terminal.

DEC VT220
As far as MS-DOS Kermit is concerned, the VT220 is identical to the VT320 except that Kermit identifies itself to the host as a VT220.

DEC VT52
An old DEC terminal with only the bare essentials.

Heath/Zenith-19
A combination of VT52 and VT102 features.

Honeywell
Identical to VT102, but Kermit identifies itself to the host as a Honeywell (now Bull) VIP-series terminal.

Tektronix 4010
A graphics terminal capable of drawing pictures on the screen. Kermit's Tektronix emulator also includes features of the 4014 model, plus much more.

Complete technical details of Kermit's terminal emulators are given in Appendix II.

Most computers, such as UNIX and VAX/VMS systems, support a variety of terminals. Certain applications on these computers expect to be able to control the appearance of your screen. Examples include EMACS or VI on UNIX, PHONE or EDT on VMS, and IBM mainframe protocol converters in general. The same is true for many dialup services, like the Digital Electronic Store that we visited earlier. For all this to work, Kermit's terminal type must agree with the host's. You must tell Kermit which type of terminal to emulate, *and* you must tell the host computer what kind of terminal Kermit is emulating. The command for selecting a terminal type is:

SET TERMINAL TYPE *name*
Specify which kind of terminal to emulate. The choices for *name* are HEATH-19, VT52, VT100, VT102, VT220, VT320, HONEYWELL, and TEK. For example:

```
MS-Kermit>set terminal type vt102
```

There is also a special type, NONE, that tells Kermit to ignore all screen formatting commands from the host.

Which Terminal Type Should I Use?

If your host supports the VT320, use that because it has the most advanced features; then, in order of preference, the VT220, VT102, Heath-19, VT100, and VT52. Use Honeywell if you are connecting to a Honeywell host. If you are using a graphics package on the host, it will most likely put Kermit into Tektronix mode automatically, but if it does not, you can set Kermit's terminal type to Tek yourself. If your host supports none of these, use NONE. Among the more advanced features of the Heath-19, VT102, VT220, and VT320 is the ability of full-screen applications to update the screen more efficiently, which is desirable on a low-speed dialup connection. Support for international character sets (see Table 13-1 in Chapter 13) is unique to the VT220 and VT320.

How Do I Tell the Host Which Kind of Terminal I Have?

The answer to this question is different for every kind of host computer. Let's look at several common cases.

VAX/VMS

First set Kermit to the desired terminal type—VT320 for release 5.0 and later of VMS, and VT220 or VT102 for older releases. Enable Xon/Xoff flow control. CONNECT to VMS and log in. VMS automatically sends an invisible "escape sequence" to ask your terminal what it is and Kermit responds automatically (see Table II-10). Now type the VMS command SHOW TERMINAL to make sure the inquiry was answered properly. Here's an example:

```
MS-Kermit>set term type vt320   (Select terminal type)
MS-Kermit>set flow xon/xoff     (Tell Kermit to do Xon/Xoff)
MS-Kermit>connect               (Start a terminal session)
                                (Press Enter key here)
Username: matt                  (Type your username)
Password: ______                (Type password; it doesn't echo)

Welcome to VAX/VMS V5.0

$ show terminal

Terminal: _TXA0:  Device_Type: VT300_Series  Owner: MATT

  Input:  9600  LFfill:  0  Width:  80  Parity: None
  Output: 9600  CRfill:  0  Page:   24
```

If your VMS terminal type is incorrect, for example if Kermit is emulating a VT320 but the VAX is running an old version of VMS, you can tell VMS your terminal type:

```
$ set terminal /device=vt200
```

The VMS command HELP SET TERMINAL /DEVICE shows a list of the terminal types available on your VAX.

DEC VT200/300 Function Keys

If the VMS system believes your terminal type is VT200 or VT300, it probably expects you to have a DEC keyboard with DEC function keys F1–F20 and other special keys like Find and Select. MS-DOS Kermit supports these keys, but does not assign their functions to PC keys automatically because it doesn't know which keys you want them assigned to. To use DEC special keys from Kermit, you must assign their functions to the desired PC keys with Kermit's SET KEY command, which is explained later in this chapter, or let the `VT300.INI` file on the MS-DOS Kermit diskette do it for you (see Appendix III):

```
MS-Kermit>take vt300.ini
```

Other VMS Considerations

When you tell VMS your terminal type is VT200 or VT300, VMS normally expects to have an 8-bit no-parity connection to your terminal. In particular, it expects to be able to send special 8-bit control characters. If you do have an 8-bit connection, be sure to give the following Kermit and VMS commands when you log in:

```
MS-Kermit>set parity none       (Tell Kermit to use no parity)
MS-Kermit>set term bytesize 8   (Tell Kermit to use 8 bits)
$ set terminal /parity=none     (Tell VMS to use no parity)
$ set terminal /eight           (Tell VMS to use 8 bits)
```

For 7-bit connections, be sure to give these commands:

```
MS-Kermit>set parity even       (Tell Kermit to use parity)
$ set terminal /parity=even     (Tell VMS the parity)
$ set terminal /noeight         (Tell VMS no 8 bit characters)
```

Also give the following commands to make sure that Xon/Xoff flow control is enabled in both directions (this applies to both 7- and 8-bit connections):

```
MS-Kermit>set flow xon/xoff     (Tell Kermit)
$ set terminal /hostsync        (VMS-to-PC flow control)
$ set terminal /ttsync          (PC-to-VMS flow control)
```

UNIX

The method for identifying your terminal to UNIX depends on which kind of UNIX system you have, which shell you are using, and how things are set up at your site. First, find out which terminal types are supported by your UNIX system, and what they are called. Each UNIX system has a different selection, stored in a database called "termcap" or "terminfo." On some UNIX systems, the available terminal types are listed in the file `/etc/termcap`. In System V based UNIX versions, this information is scattered over many files in the `usr/share/lib/terminfo` or `/usr/lib/terminfo` directory.

In general, the trick is to set your TERM environment variable to the desired terminal type, using the same name that is found in the termcap or terminfo database. These names are usually lowercase and contain no punctuation. Some typical names are vt320, vt300, vt220, vt200, vt102, vt100, h19, vt52, and ansi. Try them in that order, picking the first one you find, and set Kermit's terminal type accordingly. Use VT320 in Kermit if the UNIX name is vt300, use VT220 for vt200, and use VT100 for ansi.

Set your UNIX terminal type using the commands shown in Table 8-3, but substitute the desired terminal type for vt102. Note that case matters in UNIX commands: `VT102` is not the same as `vt102`.

You can check whether your command worked by giving the UNIX "clear" or "tput clear" command, which should clear your screen. If your screen does not clear, you have probably given a terminal name that is not in the UNIX terminal database.

Table 8-3 Setting Your Terminal Type in UNIX

Shell	*Command 1*	*Command 2*	*Check*
Bourne shell (sh)	`TERM=vt102`	`export TERM`	`echo $TERM`
C-shell (csh)	`setenv TERM vt102`	(none)	`echo $TERM`
Korn shell (ksh)	`TERM=vt102`	`export TERM`	`echo $TERM`

Some UNIX systems also support a command called "tset," which sets your terminal type as well as many other terminal characteristics:

```
eval 'tset -sQ vt102'
```

Type the UNIX command "man tset" for details. Other terminal characteristics, such as flow control and parity, are controlled with the "stty" command, whose form and options vary from one UNIX system to another. Type "man stty" for further information.

IBM Mainframes

When connecting to IBM mainframes through 3270 protocol converters such as the Series/1 or 7171, you are prompted for your terminal type. The names vary from site to site, so check your local documentation if you have difficulties:

```
ENTER TERMINAL TYPE: vt-100
```

You can usually check what terminal types are available by pressing the Enter key in response to the prompt:

```
ENTER TERMINAL TYPE:            (Press the Enter key)
VALID TYPES ARE:
ADM-3A    H19
HP2621    IBM-3101
VT-100    VT-52
ENTER TERMINAL TYPE: vt-100
```

In this case, VT100 or H19 (Heath-19) are good choices, because they have more features than the VT52, and MS-DOS Kermit does not emulate any of the others. Terminal types on the IBM mainframe must be typed in exactly; notice the `-` (dash) between the `vt` and the `100` (spelling and punctuation may differ at your site).

For IBM mainframe linemode connections, in which the host does not make any attempt to control the format of your screen, it doesn't matter which terminal type is used.

Terminal Characteristics

Besides the terminal type, there are many other terminal characteristics that you can control using MS-DOS Kermit's SET TERMINAL command. Whether you choose to use all, some, or none of the commands, you should be aware of your options. And there are so many options! Here are some examples:

SET TERMINAL WIDTH

This command tells Kermit whether to use an 80- or 132-column screen (these two numbers are the only choices, just as they are for a DEC VT100 or later terminal). The 132-column mode can be used only if your PC has a video adapter that supports it, such as the IBM XGA or various Paradise and Tseng Labs models. If Kermit knows how to command the video adapter to change screen width, it does so automatically. If it does not have built-in knowledge of the adapter, it attempts to run the DOS batch program `COLS132.BAT` for switching to 132-column mode, and `COLS80.BAT` for 80 columns. You must create these batch files and put them somewhere in your DOS PATH or your current directory. They should contain whatever command came with your video adapter to make it change widths, such as a DOS MODE command.

SET TERMINAL WRAP

The options are ON and OFF. This controls what Kermit does if it receives a line of text wider than the screen. When WRAP is ON, Kermit will break, or wrap, the line onto the next line. Otherwise, characters past the right edge of the screen will be lost. WRAP is OFF by default on the assumption that the host computer will do the line-wrapping itself. Example:

```
MS-Kermit>set term wrap on
```

SET TERMINAL TABSTOPS AT

Kermit's tabs come preset to every eight spaces, just like the DEC terminals that Kermit emulates. This command lets you change them. The operand is a list of numbers specifying where to put the tab stops, for example:

```
MS-Kermit>set terminal tabs at 10 20 30 40 50 60 70
```

You can also SET TERMINAL TABS CLEAR AT the specified numbers or SET TERMINAL TABS CLEAR ALL to remove all tab stops. If tabs are to be set at regular intervals, you can use a special notation, SET TERM TABS AT 11:10, which means set tabs every ten spaces, starting at screen column 11.

SET TERMINAL NEWLINE

Normally when you press the Enter key, Kermit sends a carriage return only. SET TERMINAL NEWLINE ON makes it send both carriage return and linefeed, which is useful for PC-to-PC communication.

SET TERMINAL DIRECTION
The options are LEFT-TO-RIGHT (the default) and RIGHT-TO-LEFT. The purpose of this command is explained in Chapter 13. For now, it's just for fun. Try it!

There are also many other terminal settings. At the `MS-Kermit>` prompt, you can type:

```
MS-Kermit>set terminal ?
```

to find out what they are. Some will be explained as we proceed. The rest of them are listed in Chapter 17.

Screen Rollback

If you use terminal emulation at all, you will appreciate this feature. How many times have you wanted to look again at something that has already left your screen? MS-DOS Kermit gives you this ability. Just press the PC's PgUp (Page Up) key to see the previous screen. Press it several times to see several previous screens. Kermit's normal retention is about ten screens, but you can change its capacity by adding a Kermit environment variable to your `AUTOEXEC.BAT` file, for example:

```
SET KERMIT=ROLLBACK 20
```

to allocate 20 screens worth of rollback memory. (Each screen requires about 4KB of memory.) After rolling back, you can roll forward again by pressing the PgDn (Page Down) key.

You can also roll back and forward a line at a time, rather than a screen at a time, by holding down the Ctrl key while you press PgUp and PgDn. And you can restore the latest (bottom, newest) screen instantly by pushing the End key. To go directly to the earliest (top, oldest) screen in Kermit's memory, press the Home key.

What should Kermit do if new characters arrive at the serial port while your screen is rolled back to some time in the past? It's your decision:

SET TERMINAL ROLLBACK ON
Restore the latest screen, and then display the new characters in their rightful place.

SET TERMINAL ROLLBACK OFF
Display new characters on the current position in the rolled-back screen. This is particularly useful if you rolled back your screen to copy something from an earlier screen.

Kermit's rollback feature is not only a convenience, it can be a lifesaver. Suppose you have been typing text into a host application—for example, a text editor or electronic mail program—for hours, when suddenly your connection to the host computer dies. Normally, all of your work would be lost. But Kermit can save you! Just press the Home key

to get to the top of your screen memory, then type *Ctrl-]F* to copy the screen to a file, press PgDn, copy the next screen, and so on to the bottom of the screen memory. All of your work is now recorded in your screen-copy file, which you can transfer back to the host (just as soon as you learn how to use Kermit to transfer files).

Session Logging

Where there is a Kermit command, there is a reason. You might someday need to record your entire session, or selected parts of it, to a PC file. You can do this by giving the LOG SESSION command. Every character that arrives at the serial port (after you give MS-DOS Kermit the CONNECT command) is recorded in the file `SESSION.LOG`. This method saves you the trouble of rolling back your screen and typing *Ctrl-]F* several times if you know ahead of time that you want this information.

You can also specify a different filename for the session log file:

```
MS-Kermit>log session who.log   (Select session logging)
MS-Kermit>connect               (Begin terminal emulation)
                                (Press the Enter key)
$                               (The other computer's prompt)
$ who am i                      (Commands are copied to WHO.LOG)
watsun!karen                    (All responses go there too)
Ctrl-]c                         (Escape back to the PC)
MS-Kermit>close session         (Stop copying screen to file)
MS-Kermit>connect               (Go back to the other computer)
```

When you are finished logging, you can close the DOS file using the Kermit command CLOSE SESSION, or you can EXIT from the MS-DOS Kermit program. Session logging can be turned off without returning to the `MS-Kermit>` prompt by typing the connect-escape sequence *Ctrl-]Q* (hold down the Ctrl key and press the] key, then press the letter Q), and it can be resumed with *Ctrl-]R*. New material is written to the end of the log. Session logging differs from screen copying with *Ctrl-]F* in several ways:

1. Screen copy saves only the current screen, whereas session logging can save an entire session or any part of it.
2. Screen copy saves only the characters that appear on the screen, but session logging saves all characters sent by the host, including screen formatting commands.
3. Screen copy saves translated characters (as they appear on the screen) whereas session logging saves untranslated characters (as they arrive from the host).

A session log that has been saved to a DOS disk can be replayed. Assuming the name of the session log is SESSION.LOG and that your terminal type was VT102 during the session, the procedure is:

```
MS-Kermit>set term vt102
MS-Kermit>replay session.log
```

You will see the entire recorded session unroll before your eyes (use the PC's Ctrl-S and Ctrl-Q keys to stop and continue the display). This can be mildly entertaining for sessions that control the screen in fancy ways, such as the Digital Electronic Store tour or a Tektronix graphics session, and can be useful too, for example, in demonstrating or marketing computer software.

Key Redefinition

Have you ever noticed that every time IBM releases a new model PC, it has changed the keyboard so much that you can't find half the keys? Do you have to retrain your fingers several times a day as you switch between your PC and some other kind of keyboard? Are there certain words or phrases that you type so frequently that you wish they could be entered magically with a single keystroke?

If you answered yes to any of these questions, MS-DOS Kermit can help. Kermit lets you assign anything you like to any key at all. These definitions are active only during terminal emulation (after you have given Kermit the CONNECT command); they do not affect the commands you type at the MS-Kermit> prompt or DOS commands.

The Kermit command to establish a key definition is SET KEY. The easiest way to use this command is to type SET KEY, and then press the Enter key. Kermit will ask you to press a key, and then it will ask you to type the new definition for that key:

```
MS-Kermit>set key
  Push key to be defined:       (You press the F1 key)
  Enter new definition: Fooey!
  Scan code \315 is defined as
  String: Fooey!
```

From now on, every time you press the F1 key, Kermit will send Fooey! to the host computer, just as if you had typed it.

You can examine a key's definition by typing the SHOW KEY command at the MS-Kermit> prompt and then pressing the desired key when MS-DOS Kermit asks you to. Practically any key or key combination can be defined: a single key, a shifted key, a Ctrl-key combination, an Alt-key combination, a Ctrl-Alt-key combination, a Ctrl-Alt-Shift combination, and so on.

Key definitions can include control characters, but only if you specify them in a particular way. You are not allowed to type them literally because they have special meanings of their own to Kermit's command processor (for example, Enter terminates a command, and *Ctrl-C* interrupts a command). This special notation is the backslash character (\) followed by a number that tells what character it is. These numbers are listed in the ASCII[9] table (Table I-5) in Appendix I. For example, ASCII character 13 is CR (Carriage Return), which is produced on the PC by pressing the Enter key. So, if you wanted to have a fast way of logging out from the host, you could put a logout command, including the carriage return, on a single key, say F2:

```
MS-Kermit>set key
  Push key to be defined:       (You push the F2 key)
  Enter new definition: logout\13
  Scan code \316 is defined as
  String: logout^M
```

The ^M tells you that Kermit has correctly interpreted your \13 as carriage return, which is indeed Ctrl-M (see Table I-5).

Notice the phrase "scan code" that appears when you type a SET KEY or SHOW KEY command. Every PC key combination has a code that identifies it. The letter A is 65, *Ctrl-A* is 1 (one), *Alt-A* is 2334, *Ctrl-Alt-A* is 3358, and so on. If you know the scan code of the key you want to define, you can write the key definition command all on one line so that further interaction is not necessary:

```
SET KEY \317 help\13
```

This assigns the word HELP followed by carriage return to the F3 key. To find out a key's scan code, type show key, and then press the desired key or key combination. For example, if you wanted to see what the scan code was for the letter a:

```
MS-Kermit>show key
Push key to be shown (? shows all): a  (You press the 'a' key)

ASCII char: a  \97 decimal is defined as
Self, no translation.
```

The example shows that the scan code for the letter a is \97.

Perhaps the most popular of all SET KEY commands is the one that moves the Esc key from wherever IBM is putting it this year (far right of the keypad on the PC/AT keyboard, extreme upper left on the PS/2 keyboard) to the location that VT100 users are accustomed to—just left of the digit 1 and above the Tab key. The key that occupies this location is

---

[9]ASCII is the American Standard Code for Information Interchange, ANSI Standard X3.4-1986. It is the code used by the PC and most other computers for representing characters.

usually the ‘ (accent grave) key. If you type the SHOW KEY command and then push the ‘ key, you will learn that its scan code is 96. And if you look up ESC in Table I-5, you'll see that its decimal value is 27. So the following command will do the trick:

```
MS-Kermit>set key \96 \27
```

But now you no longer have a way to transmit an accent grave from the keyboard. The logical answer is to assign it to the Esc key:

```
MS-Kermit>set key \324 \96
```

MS-DOS Kermit's keyboard scan codes are listed in Table I-9 in Appendix I.

But that's not all! Not only can you assign any character to any key, and not only can you assign a string of characters to any key, but you can also assign Kermit *functions* to the keys of your choice. But before you can do this, you must know the names of the Kermit functions. Table I-4 in Appendix I lists them; it also shows the default key assignments for these functions. Here's one example:

```
MS-Kermit>set key \319 \Kexit
```

This assigns the escape-back-from-connect-mode function, which is called `\Kexit` and is normally assigned to *`Ctrl-]C`* or *`Alt-X`*, to the F5 key. You can also assign multiple functions to a single keystroke, as in this example that hangs up the phone and escapes back to the MS-DOS Kermit prompt (note the placement of the braces):

```
MS-Kermit>set key \320 {\{Khangup}\{Kexit}}
```

You can even assign a mixture of functions and ordinary characters to a key, as in this example that makes the F7 key send the characters "logout" followed by a carriage return (`\13`) and then escape back to the prompt:

```
MS-Kermit>set key \321 {logout\13\{Kexit}}
```

Among Kermit's keyboard verbs you will find some that do what the function and editing keys of DEC keyboards do. If you need to use a host application that requires DEC function keys like F6–F20, Find, Insert Here, Remove, etc., you must use SET KEY commands to assign the corresponding verbs to the keys of your choice, for example:

```
MS-Kermit>set key \4434 \KdecInsert   (Insert on Insert key)
MS-Kermit>set key \4435 \KdecRemove   (Remove on Delete key)
MS-Kermit>set key \2413 \KdecF6       (DEC F6 on Alt-F6)
MS-Kermit>set key \2414 \KdecF7       (DEC F7 on Alt-F7)
```

The file `VT300.INI` on your MS-DOS Kermit distribution diskette contains assignments for all the DEC function and editing keys. To avoid typing numerous SET KEY commands every time you start Kermit, put all your key definitions into Kermit's initialization file, `MSKERMIT.INI` to make them take effect automatically each time you start Kermit. If you have different groups of key settings that you want to use at different times, collect them into separate command files and TAKE them as desired.

Screen Color

If you have a color monitor, you can use Kermit's SET TERMINAL COLOR command to pick the foreground and background colors to be used during terminal emulation. If you will be doing a lot of terminal emulation, this feature can help you find the color combination most soothing to your eyes. Moreover, if the color you choose is different from the color on your PC when you are not doing terminal emulation, it is an easy way to remember which computer you're talking to.

The operands of the SET TERMINAL COLOR command include two-digit numbers that specify the colors. A number starting with 3 gives the foreground color, and a number starting with 4 gives the background color. The second digit tells the actual color to be used (see Table 8-4), for example:

```
SET TERMINAL COLOR 34 47        (Blue on white)
```

Some of the color combinations are pretty wild—try them! If you include the number 1, you will get a high-intensity foreground, which also changes the color somewhat. Try this:

```
SET TERMINAL COLOR 1 31 45      (Purple and pink!)
```

To reset the foreground to normal intensity, include the number 0:

```
SET TERMINAL COLOR 0 31 45      (Hard to read!)
```

There are 128 possible combinations, but some of them make little sense (like black on black). If you specify the number 0 by itself, the result is white on black:

```
SET TERMINAL COLOR 0            (White on black)
```

Table 8-4 Kermit Screen Colors

Color	*Foreground*	*Background*
black	30	40
red	31	41
green	32	42
orange	33	43
blue	34	44
amethyst	35	45
turquoise	36	46
white	37	47

The SET TERMINAL SCREEN-BACKGROUND command lets you switch your screen's foreground and background colors with each other. The options are NORMAL and REVERSE. Remember that your colors can be seen only *after* you type the CONNECT command at the `MS-Kermit>` prompt. You will need other software (such as `ANSI.SYS`) for your PC to control the colors when you are not using Kermit for terminal emulation.

Printer Control During Terminal Emulation

If you have a printer attached to your PC, you can use Kermit to control it in several different ways during terminal emulation. To prevent loss of characters while printing, your printer should have a parallel (rather than serial) interface, and Kermit should have a full-duplex host connection with flow control enabled (see Table 7-1). Shared network printers generally work just like local parallel printers.

The Print Screen Key

Pressing the PrtSc (Print Screen) key (together with the Shift key on some systems such as the PC/XT) causes the current contents of the screen to be printed by DOS. This is not a Kermit function, but a DOS function. There is no Kermit keyboard verb associated with this operation, and in fact the Print Screen key cannot even be detected by Kermit.

Holding down the Ctrl key while pressing the Print Screen key invokes Kermit's `\Kprtscr` verb. The first time you press this key combination, Kermit starts copying newly appearing screen text to the printer. The next time you press it, Kermit stops copying. And so on.

Host-Controlled Printing

MS-DOS Kermit also lets the remote computer control your printer by sending special "escape sequences" that direct subsequent incoming characters to the printer (see Table II-12). There are two forms of host-controlled printing. *Transparent printing* sends all incoming characters (including escape sequences) directly to the printer without any translation or other processing, and does not put them on the screen.

Autoprint processes incoming characters normally, interpreting escape sequences and translating character sets, displays them on the screen, and then prints each screen line when the cursor leaves it.

The Mode Line

When the printer is activated during terminal emulation (except by the Print Screen key), Kermit's mode line contains the letters PRN (the DOS device name for the printer) at the far right. If the printer is not ready (for example, it is not turned on, runs out of paper, or becomes jammed), a message telling you so appears in the mode line:

```
*** PRINTER IS NOT READY *** Press R to Retry, D to Discard:
```

While Kermit waits for you to press the R or D key, you can turn on, reload, or fix the printer. Or you can just type D to cancel the printing operation altogether. Note that it can take DOS quite a long time to inform Kermit that the printer isn't printing, sometimes up to a minute.

Redirecting Your Printer

All of the printing methods just described use the DOS PRN device: the printer that is attached to your PC (either directly or on a network). You can tell Kermit to use some other device instead, or for that matter a file on disk, with this command:

SET PRINTER

This command tells Kermit to send characters that would normally go to the DOS printer device (PRN) during terminal emulation to the specified device or file. If the file already exists, new material is added to the end. Examples:

```
MS-Kermit>set printer nul              (The null device)
MS-Kermit>set printer c:\logs\x.log    (A file on disk)
```

This command applies to all the printing methods listed in this section except for the Print Screen key, which is executed directly by DOS behind Kermit's back.

If you don't have a printer, you should put SET PRINTER NUL in your `MSKERMIT.INI` file to prevent accidental printing commands from locking your PC for long periods of time.

Printing Your Session Log

You can direct your session log to the printer by specifying the device name of your printer in the LOG SESSION command, for example:

```
MS-Kermit>log session prn
```

You can temporarily stop printing by using *Ctrl-]Q* and resume it using *Ctrl-]R*, and you can close the printer's session log with the CLOSE SESSION command.

Using SET PRINTER for Session Logging

When you use SET PRINTER to redirect your printer to a file during terminal emulation, the resulting file is different from a session log because characters are sent to it only after they are formed on the screen. Thus a "printer log" contains no imbedded escape sequences. Also, if Kermit is doing character-set translation (see Chapter 13), the PC's translated characters are recorded rather than the codes sent by the host. To record a session in this manner, simply issue a SET PRINTER command specifying a filename and press Ctrl-Print Screen during terminal emulation whenever you want to start or stop recording.

Copying Your Screen to the Printer

Finally, you can set your screen copy file to be the printer :

```
Kermit-MS>set dump prn
```

Any time you want to print your current text screen, just use *Ctrl-]F*. With this option, you will get a formfeed (page eject) after each screen.

Graphics

If you are one of those people who believes that a picture paints a thousand words, then you'll like this feature. MS-DOS Kermit is capable of emulating a Tektronix 4010/4014 graphics terminal for use with host-based software that can generate Tektronix graphics images. You cannot create these graphic images with MS-DOS Kermit alone. The software on the other computer produces the graphic image—that's the graphics software. Kermit simply (actually not so simply) displays these images as the host constructs them or saves them in a file to be REPLAYed at a later time or imported into another application.

Kermit's Tektronix emulator implements a mixture of Tektronix 4010, 4014, DEC VT340, and other features to draw black and white or full color characters, lines, dots, rectangles, and fill patterns (see Tables II-24 through II-31 for a complete specification). Most popular PC graphics adapters are supported, including CGA, MCGA, EGA, VGA, XGA, AT&T/Olivetti, and Hercules. VGA and higher adapters are used in EGA mode.

Kermit tries to figure out what kind of graphics adapter you have automatically. In some cases, it might guess wrong. If it does, you can use the command SET TERMINAL GRAPHICS to tell it exactly which kind of adapter it will be using. Type the Kermit command `set terminal graphics ?` to find out which adapters are supported. The SET TERMINAL GRAPHICS command also lets you specify fore- and background colors for your graphics screen, whether you want a text cursor while in graphics mode, and whether you want text characters to erase graphic elements or overlay them. See the description of SET TERMINAL in Chapter 17.

To enter graphics mode, give the command SET TERMINAL TYPE TEK, or let your host application put Kermit into graphics mode automatically by sending a special escape sequence (see Table II-24).

If you escape back to the `MS-Kermit>` prompt and then reconnect, your graphics screen image is restored if your graphics adapter had sufficient memory to store it. This way, Kermit can give you the illusion of having two windows—one with text and one with graphics—but only one window can be on your screen at a time.

```
MS-Kermit>set terminal graphics EGA
MS-Kermit>set terminal type tek
MS-Kermit>connect

THIS PICTURE INTENTIONALLY LEFT BLANK

Alt-X
MS-Kermit>set terminal type vt100
MS-Kermit>connect

Here is my text screen again.
```

Once in graphics mode, you can get your text-mode screen back quickly by typing Kermit's "toggle terminal type" key(s), normally *Alt--* (hold down the Alt key and press the minus key). The text screen is preserved for you during graphics operation. You can jump back to your graphics screen by typing *Alt--* again, or by escaping back and giving the SET TERMINAL TYPE TEK command, or the host can send an escape sequence to restore your text screen. You can clear your current screen during CONNECT mode by using *Alt-=* (hold down the Alt key and press the = key).

If you hear a beep during Tektronix emulation, that means the host has sent picture elements that are outside the current screen boundaries. Look at the current picture until you're tired of it, and then press the Enter key to see the new material. The old picture will disappear from your screen, and the new picture is displayed.

During your graphics session, a cross-hair cursor that looks like a large "+" (plus sign) might appear. This means that the host application wants you to move the cross hairs to some point on the picture and then press the Enter key. This sends the screen coordinates of the cursor to the host application. Move the cross-hair cursor using your keypad arrow keys, shifted for coarse movements and unshifted for fine tuning. If your PC has a mouse, you can use it to move the cross-hair cursor (Kermit contains built-in support for Microsoft-style mice; for others you need an external mouse driver). Push the right mouse button to send the cross hair's position to the host.

Kermit's screen copy feature works in a special way with graphics screens because the screen contains a graphics image rather than text characters and each point on the screen has color, brightness, and other attributes. When you copy a graphics screen, Kermit saves it in Aldus/Microsoft TIFF (Tagged Image File Format) 5.0 (approximately 110K per EGA screen), suitable for importation into applications such as WordPerfect 5.0, Aldus Pagemaker, Ventura Publisher, PC Paint, and others that support TIFF 5.0. Each graphics screen is saved in a separate file. The file names are `TEKPLT01.TIF`, `TEKPLT02.TIF`, and so on. You cannot use *Ctrl-]F* to dump a graphics screen; use *Ctrl-End* instead.

The graphics commands that the host transmits to create the screen image can also be recorded on your DOS disk by using Kermit's LOG SESSION command, and the resulting graphic images can be recreated by using Kermit's REPLAY command on the session log file:

```
MS-Kermit>log sess tek.log      (Record session in file TEK.LOG)
MS-Kermit>set term type tek     (Set terminal type to Tektronix)
MS-Kermit>connect               (Begin terminal emulation)

(Create graphics screen)

Alt-X                           (Hold down the Alt key and press X)
MS-Kermit>close sess            (Close the session log)
MS-Kermit>replay tek.log        (View the graphics screen again)
```

If the session log contains an escape sequence that causes Kermit to switch automatically to graphics mode (Table II-24), the SET TERM TYPE TEK command is not necessary (but it won't hurt).

The connect-mode printer control keys do not normally work with graphics. To print graphics screens, you need a special print driver program, such as the `GRAPHICS.COM` file supplied by IBM with DOS, which works with CGA systems and IBM printers. To get a sensible printout, be sure to set your graphics screen background color to black. The foreground color doesn't matter, as long as it is not also black.

Despite its length, this chapter has only scratched the surface of Kermit's terminal emulator. A few additional features will be presented later on, but for a detailed summary see the SET TERMINAL description in Chapter 17 and the escape sequence listings in Appendix II.

Chapter 9

File Transfer

The time has finally come to begin transferring files with Kermit. This is probably what attracted you to the MS-DOS Kermit program in the first place, and rightly so. If you followed along in the preceding chapters you should be connecting and interacting comfortably with any computer that you have access to, so file transfer should be a snap.

Suppose you are one of the new breed of "telecommuters" who works at home on a PC and stays in touch with the office through a modem. You could remain connected to the corporate mainframe all day and use its facilities to do your work. But that would be hard on your pocketbook at phone bill time. Or you could prepare your reports, messages, calculations, analyses, and so on, on your PC, and then transmit them to the mainframe when they are done. This approach not only cuts phone costs and keeps your phone from being tied up all day, it also alleviates the load on the mainframe and eliminates the time wasted when phone connections drop unexpectedly and you lose your work.

This is only one scenario among the thousands imaginable. If you are reading this book, you have your own reasons for wanting to transfer files. So let's get on with it.

Transferring Text Files

Begin by using all the familiar steps to start Kermit on the PC: Set the appropriate communication parameters (see Table 7-1), CONNECT, dial the host (other) computer with your modem, and log in with your access codes.

Figure 9-1 Kermit to Kermit

```
C>                                  (The DOS prompt on your hard disk)
C>kermit                            (Start up Kermit on your PC)
IBM PC MS-Kermit V3.11
Copyright (C) Trustees of Columbia University 1982,1991
Type ? or HELP for help
MS-Kermit>connect                   (Begin terminal emulation)

Welcome to VAX/VMS ...

Username: sal                       (Log in)
Password: ======                    (Type password; it doesn't echo)
```

Now comes the new part. To transfer your file, you must run a *second* Kermit program on the host computer. *Both* computers must be running a Kermit program. Otherwise, one side will not know the rules and cannot play the game correctly. One computer has to know how to send the file, and the other must know how to receive it (see Figure 9-1). Luckily, there are Kermit programs available for almost any computer you can think of, and the cost is nominal.

Most host Kermit programs are very similar to MS-DOS Kermit. Usually, you start them by typing `kermit`, and you get a prompt, like `C-Kermit>` or `Kermit-CMS>` or `Kermit-TSO>` or `Kermit-32>`. This helps remind you which Kermit program you are talking to. The basic commands of most Kermit programs are the same: HELP, SET, SEND, RECEIVE, and so forth. Type `help` and read the host Kermit's help message. Type `exit` to quit the host Kermit and return to the host's main system prompt.

```
$ kermit                          (Run VAX/VMS Kermit)

C-Kermit, 5A(166) 8 Feb 91        (The alien Kermit's herald)
Type ? for help                   (and greeting)

C-Kermit>                         (and prompt)
C-Kermit>exit                     (Issue the EXIT command)
$                                 (Back at the VAX prompt)
```

Each different Kermit program has its own peculiarities and documentation to describe its commands, just as MS-DOS Kermit has this book. Some host Kermits (such as C-Kermit for UNIX and VAX/VMS) provide "menu on demand" when you type a question mark, just like MS-DOS Kermit. Certain others, such as IBM mainframe Kermit, require you to press Enter after the question mark. But almost all Kermit programs have built-in help of some kind.

Take some time to familiarize yourself with the host Kermit program. If you are completely baffled, read the documentation. If you can't find documentation, ask the host system administrator, or contact Kermit Distribution (see the Preface).

In the following examples, you will see different host systems—VAX/VMS, UNIX, and IBM mainframe operating systems like MVS/TSO and VM/CMS. Basic use of Kermit with these systems is the same, differing mainly in the communication parameters you must set to use them successfully. We already saw how to do this in Chapter 7.

The SEND and RECEIVE Commands

The basic commands for file transfer are pretty much what you might expect, SEND and RECEIVE. The SEND command sends the named file or files to the other Kermit program, which must be given a RECEIVE command. The RECEIVE command (you can probably finish this sentence yourself) waits for a file to arrive from the other Kermit, which must be given a SEND command. You have to issue the SEND or RECEIVE command to the remote Kermit first and then escape back (return) to MS-DOS Kermit and issue the corresponding RECEIVE or SEND command.

In the examples, the character for escaping back from the host computer to the PC is shown as *Alt-X*. You can also use *Ctrl-]C*. The two are interchangeable. Use whichever one best matches your typing habits, or use SET KEY to assign this function (`\Kexit`) to any other key of your choice; for example:

```
MS-Kermit>set key \315 \kexit
```

to put it on the F1 key (see Chapter 8).

Uploading Files: The SEND Command

Sending files from your PC to a remote computer is called *uploading*. The procedure for uploading a file from the PC is:

1. Start Kermit on your PC.
2. CONNECT to the remote computer, log in if necessary, and then start Kermit there.
3. Type the RECEIVE command to the host Kermit.
4. Escape back to the PC's `MS-Kermit>` prompt.
5. Type the SEND command, specifying the name of the PC file to be sent.
6. When the transfer is complete, CONNECT back to the host system and then EXIT from the host Kermit.

SEND is one of Kermit's most important commands. It is used so often that it has a special abbreviation, S, even though many of Kermit's other commands also start with S.

Remember to log out from the host when you are finished using it. If you have dialed the host with a modem, logging out should also hang up your phone connection automatically. Watch your modem lights to make sure.[10] If the lights are still on after you log out, your telephone connection is still active. You can hang it up with Kermit's HANGUP command. If all else fails, turn off your modem.

Now let's run through a real example. You have a PC, a hard disk, a Hayes or Hayes-compatible modem,[11] and a Touch-tone telephone, and the remote host is a UNIX system. The file you want to upload from your PC to the UNIX computer is a text file (all printable characters) and is stored on your PC in the directory `\REPORTS` with the filename `REPORT.TXT`. Let's take it from the very beginning, after you have turned on your PC:

```
C>                                  (The DOS prompt on your hard disk)
C>kermit                            (Start up Kermit on your PC)
IBM PC MS-Kermit V3.11
Copyright (C) Trustees of Columbia University 1982,1991
Type ? or HELP for help

MS-Kermit>cd \reports               (Change directory to \REPORTS)
MS-Kermit>set speed 1200            (Set speed to match modem)
```

[10] Most external modems have status lights. Internal modems do not. You can find out the status of an internal modem by using Kermit's SHOW MODEM command.

[11] If you are using a different kind of modem, the dialing procedure and modem responses (like `CONNECT 1200`) may be different. If you are using a "dumb modem," you must dial manually.

```
MS-Kermit>connect              (Begin terminal emulation)
AT                             (Get attention of Hayes modem)
OK                             (Modem says "I'm ready")
ATDT 212 555-4321              (Type Hayes dialing command)
rrrriiinnnngggg                (Phone rings and picks up)
CONNECT 1200                   (The other computer answers)

Ultrix V2.0 (cunixc), tty34
8 users, load average: 0.42 0.42 0.59

login: phyllis                 (Type your username)
password:______                (Type your password; it won't echo)
$                              (The UNIX computer prompt)
$ kermit                       (Run UNIX Kermit)

C-Kermit, 5A(166) 8 Feb 91, VAX/Ultrix
Type ? for help

C-Kermit>receive               (Tell UNIX Kermit to receive a file)
Escape back to your local system and give a SEND command...

Alt-X                          (Hold down the Alt key and press X)
                               (to escape back to the PC)

MS-Kermit>send report.txt      (Tell PC Kermit to send the file)

      (The file is transferred...)

MS-Kermit>                     (All done; prompt reappears)

MS-Kermit>connect              (Connect to the UNIX computer again)
C-Kermit>exit                  (Exit from UNIX Kermit)
$ exit                         (Log out from UNIX)
Alt-X                          (Hold down the Alt key and press X)
                               (to get back to the PC)

MS-Kermit>exit                 (Return to DOS)
```

Congratulations, you have just transferred your first file! You can believe Kermit's `Sending: Complete` message, or you can simply look to see that the filename appears on the receiving computer. If you are still skeptical, you can TYPE or PRINT the newly received file and compare it with the original. And, if you *still* aren't convinced, you can transfer the file back to the PC under a different name and use the DOS command COMP to compare the transferred file with the original file:

```
C>comp report.txt report.new
```

Sending Multiple Files

MS-DOS Kermit can also send a group of files with a single command if you include one or more wildcard characters in the SEND command filename; for example:

```
MS-Kermit>send s*.txt
```

sends all files in the current directory whose names start with S and that have a filetype `.TXT`. MS-DOS Kermit wildcards behave exactly like DOS wildcards, except that you can't use a question mark in the first position of a filename because the question mark gives help in that position; use the # character instead:

```
MS-Kermit>send #of?.txt
```

This command will send all files of type `.TXT` whose names are exactly four characters long with `OF` as the second and third characters, such as `OOFA.TXT` and `TOFU.TXT`.

File Transfer Display

Notice the display on the screen while the file is being transferred. This gives you an up-to-the-minute status report of the progress of the file transfer:

```
          MS-Kermit: V3.11

          File name: REPORT.TXT
 KBytes transferred: 7
Percent transferred: 35%
           Sending: In progress

  Number of packets: 74
      Packet length: 93
  Number of retries: 0
         Last error: None
       Last message:

X: cancel file, Z: cancel group,
E: exit nicely, C: exit abruptly, Enter: retry
```

The line `Sending: In progress` will change to `Sending: Complete` when the file has been transferred successfully. If the file transfer failed for any reason, this line will say `Sending: Failed`, and the `Last error` field will display the reason, for instance `Insufficient disk space`.

The file transfer display fields should be self-explanatory. `KBytes transferred` means (approximately) how many thousands of characters have been transferred so far. The packet statistics are updated continuously. If you should see them stop, you know something is wrong. Try pressing the Enter key once or twice if this happens.

Several other file transfer display options are available. The command to select them is SET FILE DISPLAY. The options are REGULAR (the display shown above), QUIET (no display at all), and SERIAL (see Chapter 15):

```
MS-Kermit>set file display ?  One of the following:
 Quiet, Regular, Serial
MS-Kermit>set display quiet
```

File Transfer Interruption

The bottom line of the file transfer display shows the options you have for interrupting the file transfer:

X Press the X key to stop the current file. The portion of the file transmitted so far is not kept by the receiving Kermit unless it was given the command SET INCOMPLETE KEEP prior to the file transfer. If multiple files are being transferred, the next file will begin.

Z Press the Z key to stop the current file (just like pressing the X key). No more files will be sent, and MS-DOS Kermit will return to its prompt.

E Press the E key if X or Z didn't work.

C Press the C key if E didn't work.

Q Send a Ctrl-Q (XON) to try to break a flow control deadlock.

Enter If nothing is happening on the screen and the file transfer seems to be stuck, press the Enter key to wake it up.

To illustrate, suppose you have given the command:

```
MS-Kermit>send *.*
```

to send all of your files. As you sit in your comfy chair watching the file transfer display, you see the filenames change as one file after the other is successfully transferred. But suddenly you see the name `HUMONGUS.TXT`. This was not one of the files you meant to transfer, and it's so huge that it will take hours for Kermit to finish with it and get on to the next file. This is a good time to use the X key.

Again, suppose you have given the command:

```
MS-Kermit>send *.*
```

This time as you watch the filenames go by, it slowly dawns on you that these are not the files you meant to send. You were CD'd to the wrong directory! Time to bail out: Press the Z key, and Kermit will stop. You will still have to delete whatever unwanted files arrived at the other end.

Sending a File under an Assumed Name

If, for reasons known best to yourself, you want to upload a file from your PC under a filename different from the one used on the PC, Kermit will let you change the file's name in flight. Just include the new name in the SEND command, after the original name:

```
MS-Kermit>send whoopee.txt serious.doc
```

The PC file `WHOOPEE.TXT` will be stored at its destination as `SERIOUS.DOC`.

Downloading Files: The RECEIVE Command

Receiving files at your PC from a remote computer is called *downloading*. The procedure for downloading a file to the PC is as follows:

1. Start Kermit on your PC.
2. CONNECT to the remote computer, log in if necessary, and then start Kermit there.
3. Type the SEND command to the host Kermit, specifying the name of the file to be sent.
4. Escape back to the PC's `MS-Kermit>` prompt.
5. Type the RECEIVE command.
6. When the transfer is complete, CONNECT back to the host system, and then EXIT from host Kermit.

Because the RECEIVE command is so fundamental to Kermit's operation, it can be abbreviated simply R, even though several other Kermit commands start with the same letter.

Like the SEND command, the RECEIVE command can include an optional new name to store the file under when it arrives:

```
C-Kermit>send geo.doc            (Send a file from UNIX)
Alt-X                            (Hold down the Alt key and press)
                                 (the X key to get back to the PC)
MS-Kermit>rec whatsup.doc        (Receive with new name)
```

This will store the UNIX file `geo.doc` as `WHATSUP.DOC` on the PC. You can also specify a device and/or directory to have the file stored there under its own name or under any name you specify:

```
MS-Kermit>receive c:\bugs
```

Most modern versions of Kermit, when sending a file to MS-DOS Kermit, will include advance notice of the file's size. If a file is bigger than the available space on the current disk, MS-DOS Kermit will refuse it. This feature can save you a lot of time and phone charges—imagine having the file transfer fail after receiving 9.9 megabytes of a 10 megabyte file over a long-distance dialup connection at 1200 bps! (You can do the math yourself.)

This advance notice comes in an *attribute packet*, which also may include some other information about the file, such as its creation date. MS-DOS Kermit will use this date when creating the new file on the MS-DOS disk. So if you are a last-minute worker, you cannot pretend you created your file a week ago and Kermit changed its date to today.

If MS-DOS Kermit's treatment of attribute packets bothers you, you can disable it by giving the command SET ATTRIBUTES OFF. Disabling this feature means that no size information will be announced, you will not see what percentage of the file has been received, and received files will be stored with the PC's current date and time.

Now let's try downloading a text file called MEETINGS.TXT from a VAX/VMS computer to the PC. This procedure differs from the previous one only in exchanging the SEND and RECEIVE commands and in the details of using the remote host:

```
C>                                  (The DOS prompt on your hard disk)
C>kermit                            (Start up Kermit on your PC)

IBM PC MS-Kermit V3.11 ...
Copyright (C) Trustees of Columbia University 1982,1991
Type ? or HELP for help

MS-Kermit>set speed 1200            (Set speed to match modem)

AT                                  (Get modem's attention)
OK
ATDT2127654321                      (Type Hayes dialing command)
rrrriiinnnngggg                     (Phone rings and answers)
CONNECT 1200                        (Modem confirms)

Welcome to VAX/VMS...
Username: roberta                   (Put in your username)
Password: ________                  (Type your password)
$                                   (The VMS system prompt)
$ kermit                            (Run Kermit on the VAX)

C-Kermit, 5A(166) 8 Feb 91, VAX/VMS
Type ? for help

C-Kermit>set default [.reports]     (Change directory)
C-Kermit>send meetings.txt          (Tell VAX Kermit to send the file)

Alt-X                               (Hold down the Alt key and press X)
                                    (to escape back to the PC)

MS-Kermit>receive                   (Tell PC Kermit to receive a file)

      (The file is transferred...)

MS-Kermit>                          (All done; prompt reappears)

MS-Kermit>connect                   (Connect to VMS again)
C-Kermit>exit                       (Exit from VMS Kermit)
$ logout                            (Log out from VMS)
Alt-X                               (Escape back to the PC)

MS-Kermit>exit                      (Return to DOS)
C>
```

That's all there is to it. Now you know how to send and receive files with Kermit. But the book does not end here. That's because neither do the capabilities and options of the MS-DOS Kermit program. You don't have to be a passive receiver. Just like an MS-DOS Kermit receiver, you have options too.

Interrupting File Reception

Suppose that you are receiving files and you want to stop the file transfer. The file interruption keys X, Z, E, C, Q, and Enter work the same way as when sending the file. You have given the command:

```
MS-Kermit>receive
```

and the other Kermit is sending a file or files that you don't want. You pressed the X key, and it didn't work—the file kept coming. So you tried the Z key, but that didn't work either. (This is hypothetical; usually these will work.) At this point, use the E key. That should put a stop to it.

Filename Collisions

Kermit renames incoming files for you automatically if the name of the incoming file is the same as the name of an existing PC file:

```
          MS-Kermit: V3.11

          File name: MEETING.TXT as MEETING1.TXT
 KBytes transferred: 7
Percent transferred: 35%
          Receiving: In progress

  Number of packets: 74
      Packet length: 93
  Number of retries: 0
         Last error: None
       Last message: Renaming file to MEETING1.TXT
```

This safety feature of MS-DOS Kermit is activated automatically. You can change Kermit's file collision action with the command:

SET FILE COLLISION

Tell Kermit what to do when a file arrives that has the same name as an existing file. The options are RENAME (rename the incoming file, the default action), OVERWITE (let the incoming file overwrite existing files of the same name), and DISCARD (refuse to accept an incoming file that has the same name as an existing file, a handy option for resuming an interrupted wildcard transfer where it left off):

```
MS-Kermit>set file collision ?
 Overwrite  Rename  Discard
MS-Kermit>set file collision discard
```

Text versus Binary Files

A *text* file contains all printable letters, digits, and symbols that you can type and read on the screen. Text files can be transferred between any two Kermit programs. The Kermit programs know how text files are supposed to look on each computer and adjust the format, if necessary, to make these files usable after transfer. This way you can type the file, edit it, or whatever else you need to do as if you had originally created it on the computer it was transferred to. Examples of text files include:

`AUTOEXEC.BAT`	*Your DOS startup file*
`MSKERMIT.INI`	*MS-DOS Kermit's initialization file*
`KERMIT.HLP`	*Help file for MS-DOS Kermit*

A *binary* file contains information intended for the computer, not for you. Binary files are specific to a particular application or a particular kind of computer or device. Not only must you tell Kermit that the file should not be changed during file transfer, but usually you will not be able to use this file on a different type of computer. You may still want to do binary transfer to store programs on a system that cannot use them so that others can download them at a later time. Examples of binary files include:

`COMMAND.COM`	*The MS-DOS command processor*
`KERMIT.EXE`	*The MS-DOS Kermit program*
`BUDGET.WKS`	*A Lotus 1-2-3 spreadsheet*
`PHONES.DBF`	*A dBase database file*

With the sophistication of word processors, the distinction between text and binary is not always as clear as it seems. A word processor program that allows you to save a file with underlining and **bold** headings, for example, has inserted some control information in the file, as well as the text you typed, so when you view or print the file from within the word processor, it can remember where to underline and boldface.

Since this control information is not text, the file must be categorized as binary to be transferred correctly. If this type of file is transferred to another type of computer, it cannot be used there in any normal way because the other computer won't know what to do with the control characters that were inserted unless you have an application on the remote computer that understands the file's format. If there is a way for you to save the file as "text-only" or "ASCII-only" (most word processors let you do this, see page 25), all the special formatting controls are discarded, and you should be able to transfer this file in usable form to another computer, but without the underline, **boldface**, *italics*, and other special effects.

In general, neither the PC nor Kermit can tell whether a file is text or binary. Only you (and your hairdresser) know for sure.

Transferring Binary Files

Transferring binary files is the same as transferring text files except that you have to give the command SET FILE TYPE BINARY to *both* Kermit programs.

Remember that unless you are transferring a binary file between two computers that are of the same type (like two IBM PCs), you usually cannot use the binary file on the computer you send it to. But if you transfer a binary file from the IBM PC to, say, an IBM mainframe and then from the IBM mainframe to another IBM PC, you can use the file on the other IBM PC as long as you remember to SET FILE TYPE BINARY each step of the way.

```
MS-Kermit>connect                 (Connect to the host computer)
.kermit                           (Start Kermit on the host)
Kermit-CMS>set file type binary   (The important command)
Kermit-CMS>receive                (Tell it to receive)
Alt-X                             (Hold down Alt key and press X)
MS-Kermit>set file type binary    (Here too!)
MS-Kermit>send kermit.exe         (Send the binary file)

      (The file is transferred...)

MS-Kermit>                        (Finished; prompt reappears)
```

Unless you are transferring files between two PCs, *don't mix text and binary files* in the same SEND command. Most Kermit programs can be in only one file mode at a time, text or binary.

Transferring Files with IBM Mainframes

When transferring files with IBM mainframes, make sure to set the appropriate communication parameters before connecting (see Chapter 7). To recap, if you have a *fullscreen* connection, you can usually use all of MS-DOS Kermit's default parameters, except you probably have to use some kind of parity, and you have to set your terminal type:

```
MS-Kermit>set parity even         (Use even parity)
MS-Kermit>set terminal vt100      (Set terminal type)
```

For a *linemode* connection (line at a time, no screen control), you have to change most of MS-DOS Kermit's defaults:

```
MS-Kermit>set parity mark         (This one may vary)
MS-Kermit>set flow none
MS-Kermit>set handshake xon       (This one too)
MS-Kermit>set duplex half
```

Aside from the different communication parameter settings, you can use Kermit with an IBM mainframe just as you would any other kind of host.

Trouble

Maybe "it's not easy being green," but it should be easy to transfer files with Kermit. And we will see that it will get even easier. The hard part is figuring out what you did wrong when file transfer doesn't work.

First, did you set all the communication parameters correctly before connecting to the other computer? You look confused. Did you skip the earlier chapters? Tsk tsk. Go back and read them (especially Chapter 7) to see how you were supposed to prepare for file transfer. Next, issue the MS-DOS Kermit SHOW COMMUNICATIONS command, and look for any obviously incorrect settings (parity, flow control, handshake). Fix them with SET commands, and then try again. If you are able to connect successfully, but file transfer still doesn't work, consider:

The Obvious

Did you remember to start Kermit on the other computer? Did you remember to give it a SEND or RECEIVE command before escaping back to the PC?

Parity

The most common cause of file transfer failure is parity. Kermit would prefer not to use parity, but if the host computer or the network between your PC and the host uses it, you should tell Kermit about it; otherwise, the unexpected addition of parity bits could cause Kermit's error checking to fail. Solution: SET PARITY EVEN (or whatever the parity really is). Give this command to *both* Kermit programs.

Duplex

If you are connected to a half-duplex computer system, you should have given the command SET DUPLEX HALF. But this command has several functions and you might not want them all: It enables local echo (equivalent to SET LOCAL-ECHO ON), it turns off full-duplex software flow control (equivalent to SET FLOW NONE), and it enables RTS/CTS hardware communication line access control. It is possible that RTS/CTS is causing the problem. To find out, try this :

```
MS-Kermit>set duplex full
MS-Kermit>set local-echo on
MS-Kermit>set flow none
```

Handshake

For transferring files with half-duplex computer systems, the normal line turnaround handshake character is XON. Did you give the command SET HANDSHAKE XON? Your host computer may use a different character, or none at all, for this purpose. Consult your host system documentation or administrator to find out, or experiment with different values for SET HANDSHAKE.

Adapting to Communication Line Noise

Suppose you have a direct, high-speed connection to your company's mainframe computer at work, but from home you use your modem and regular telephone lines to access the same computer. The regular telephone lines are prone to static or clicking noises, but the direct line is far more "clean"—free from this kind of interference.

When a connection is more apt to have noise interference, there are a few parameters in the MS-DOS Kermit program that can help prevent file transfers from failing:

SET RECEIVE PACKET-LENGTH
: The messages into which Kermit breaks up your file are called packets, normally 94 characters long. When the connection is noisy, reducing the number of characters sent in a single shot increases the chance that they will all make it to the other side intact and decreases the cost of packet retransmission.

SET RETRY
: The Kermit program will try to send a particular packet a certain number of times (five, for example) before it gives up. This is how Kermit detects a broken connection. But under noisy conditions, the fact that a packet has been retried many times need not mean the connection has dropped. Under these conditions, you can increase this number to prevent unnecessary failures.

SET BLOCK-CHECK
: No error detection mechanism is foolproof. Under noisy conditions, it's more likely (but still *very* unlikely) that a transmission error could slip through undetected. To decrease the chance of this happening, you can select a more powerful detection method. There are three levels: 1 (single-character checksum), 2 (2-character checksum), and 3 (3-character 16-bit CRC). The higher the level, the stronger the error detection, but also the higher the price paid in transmission and computing overhead. If you do not say otherwise, 1 is used.

In the following example, we instruct Kermit to be more persistent under noisy conditions by allowing more retries, using smaller packets, and using the strongest of Kermit's error-checking techniques:

```
MS-Kermit>set retry 20
MS-Kermit>set packet-length 40
MS-Kermit>set block-check 3
```

Improving Kermit's Performance

Let us hope that by now you have Kermit transferring files correctly and completely. If you had problems, you've very likely solved them. It may have taken some experimentation and even some reading, but the fact is that Kermit can transfer files in almost any

environment with almost any kind of computer—a claim that no other file transfer protocol can make. But the price for this universality is paid in efficiency. Now let's look at some ways of making file transfer go faster.

Packet Length

One factor in Kermit's success is its use of relatively short packets: The normal packet length is 94. These short packets are able to slip through all sorts of narrow openings in which longer packets would get stuck. But if you know that your connection will allow long bursts of data, you can increase MS-DOS Kermit's packet length to any number up to 2000. The longer the packet, the higher the ratio of real information to protocol overhead, and therefore the greater the efficiency of the file transfer. But to use long packets, *both* Kermit programs must support this feature.

The command that controls the packet length is SET RECEIVE PACKET-LENGTH. This is the very same command that helped us adjust to noisy communication lines by making the packets smaller. The Kermit program that is receiving files controls the packet length, so give this command to MS-DOS Kermit if you are downloading, and give it to the host Kermit if you are uploading. If you plan to send files in both directions, give it to both Kermits:

```
MS-Kermit>set receive packet-length 1000
MS-Kermit>connect
C-Kermit>set receive packet-length 1000
C-Kermit>
```

(Remember, Kermit commands can be abbreviated. Normally, all you would have to type here is SET REC PAC 1000.) When you transfer a file using long packets, watch the file transfer display. The `Number of packets` field will change more slowly because there is more data in each packet, as you can see by looking at the `Packet length` field:

```
          MS-Kermit: V3.11

          File name: REPORT.TXT
 KBytes transferred: 7
Percent transferred: 35%
            Sending: In progress

  Number of packets: 7
      Packet length: 1000
  Number of retries: 0
         Last error: None
       Last message:
```

A big advantage of long packets is that they can be used on either full-duplex or half-duplex connections. But there are also several risks. First, long packets may trigger the very problem that Kermit's regular short packets were designed to avoid: buffer overflows and the resulting data loss. If 94-character packets work for you, but 1000-character packets consistently fail, try to home in on the largest size that works.

Second, Kermit's normal error-checking technique is really not strong enough for very long packets. So when using long packets, you should also SET BLOCK-CHECK 2 or 3.

Third, if the connection is noisy, long packets may actually *reduce* your efficiency rather than increase it. That's because the longer the packet, the greater the chance it will be damaged under noisy conditions, and the more time it takes to retransmit it.

You can use Kermit's SHOW STATISTICS command to measure your file transfer efficiency. Using the same transmission speed, compare the file characters per second for long packets with the file characters per second for regular packets.

Sliding Windows

Normally, the sending Kermit program transmits a packet (message) and then waits for the receiving Kermit program to send a response back saying "I got it." The sending Kermit program then transmits another packet. Kermit's "stop-and-wait" style of packet exchange works on both full- and half-duplex connections. But stopping and waiting for a reply can be costly on long-distance connections, like those through earth satellites or public data networks.

On full-duplex connections, where both computers can send and receive data simultaneously, it is not necessary for each Kermit to be silent while the other one is transmitting. The delay caused by waiting for a reply to each packet can be eliminated if we allow a certain number of packets to be sent without replies and let the replies come later. This "certain number" is called the *window size*. As long as the first reply comes before the window size is exceeded, transmission of packets can be continuous, even over long-delay connections. Kermit's window size can be from 1 to 31:

```
MS-Kermit>set window 8
```

It is best to use windows with regular (short) packet lengths of 94 or fewer characters. This way, loss of performance is minimized when packets are damaged by noise. On a very clean connection, however, you can combine sliding windows and long packets:

```
MS-Kermit>set window 8
MS-Kermit>set receive packet-length 250
```

The window size (number) you request is the maximum number Kermit will use. The actual number of window slots in use at any instant shows up in the file transfer display:

```
          MS-Kermit: V3.11

          File name: REPORT.TXT
 KBytes transferred: 14
Percent transferred: 70%
            Sending: In progress
Window slots in use: 3 of 4
```

```
Number of packets: 140
    Packet length: 94
Number of retries: 0
       Last error: None
     Last message:
```

This number is determined by various factors, including the amount of delay from one end to the other and the amount of noise on the connection. The longer the delay, or the worse the noise, the more window slots need to be used. When MS-DOS Kermit is receiving files, it normally reports its window size as 1 even if the file sender is using a larger window size. That's because MS-DOS Kermit never fills up its receive window if all packets arrive in order. The receiver's window only comes into play when packets are lost due to transmission errors.

Sliding windows can be used only with other Kermit programs that support this feature, such as PRIME Kermit or C-Kermit 5A or later on UNIX and VAX/VMS. If you try to use sliding windows with a Kermit program that does not have this feature, the two Kermits will automatically use the stop-and-wait method instead.

Compression

MS-DOS Kermit can also compress your file during transmission. If a particular character occurs more than twice in a row, a special repeat-count prefixes a single copy of the character. MS-DOS Kermit automatically negotiates the compression feature with the other Kermit—no SET commands required. This compression method can be surprisingly effective because text files often contain repeated space characters and binary files often contain large runs of zero bytes.

Chapter 10

Using a Kermit Server

Now that you are an experienced Kermit user, you are ready to move up to a new level of convenience. No longer must you continuously escape back and forth between MS-DOS Kermit on the PC and the Kermit program on the other computer if you are transferring a lot of files whose names are not similar enough for wildcards or if you are sending some files and receiving some others. Instead, you can connect to the other computer and start up the Kermit program there as before, but this time you can put the host Kermit into *server mode* by issuing the SERVER command.

Once the remote Kermit is in server mode, you only need to type commands to the PC's `MS-Kermit>` prompt to send and get files. MS-DOS Kermit will relay any commands intended for the other computer to the remote Kermit server automatically. In the following example, we use a Hayes (or Hayes-compatible) modem to dial up an IBM mainframe, set the mainframe Kermit to be a server, and transfer files to the PC and from the PC.

```
C>                                  (The DOS prompt)
C>kermit                            (Start up Kermit on your PC)

IBM PC MS-Kermit V3.11 ...
Type ? or HELP for help

MS-Kermit>ibm                       (See Chapter 14)
MS-Kermit>set speed 1200            (Set the speed)

AATT                                (Half duplex, double echo)
OK                                  (Modem is ready)
ATDT2127654321                      (Issue the dial command)
```

```
rrrriiinnnngggg                 (Phone rings and answers)
CONNECT 1200                    (Modem confirms)

VIRTUAL MACHINE/SYSTEM PRODUCT
!
.login louie                    (Type your username)
Enter password: _______         (Put in your password)
.                               (The VM/CMS prompt is a period)
Ready; T=0.07/0.11 11:58:28
CMS
.kermit                         (Run mainframe Kermit)
Kermit-CMS Version 4.2
Enter ? for a list of valid commands

Kermit-CMS>server               (Put it in server mode)
Entering server mode.  Please escape to
local Kermit now.  To terminate the server
use the BYE or FINISH commands.

Alt-X                           (Escape back to your PC)

MS-Kermit>send oofa.*           (Send all of my OOFA files)

    (The files are transferred...)

MS-Kermit>get meeting.txt       (Tell server to send a file)

    (The file is transferred...)

MS-Kermit>                      (Done; prompt reappears)
MS-Kermit>connect               (Go back to the IBM mainframe)
```

As you can see, you can send and receive as many files as you like without having to escape back and reconnect.

If you do reconnect, you will see the screen as you left it when you last escaped back to MS-DOS Kermit. When you try to type to the IBM mainframe directly, nothing happens. This is because the mainframe Kermit is still in server mode. You can communicate with it only by typing commands in response to the `MS-Kermit>` prompt.

When you are done, you can give the BYE command. This shuts down the remote Kermit server and logs out your host session. In this case, there is no need to reconnect.

```
MS-Kermit>bye                   (Terminate mainframe session)
C>                              (BYE also exits to DOS)
```

If you want to shut down the server and log out the host session but remain in MS-DOS Kermit, use the LOGOUT command instead:

```
MS-Kermit>logout                (Terminate mainframe session)
MS-Kermit>                      (Back to MS-Kermit prompt)
```

If you want to continue terminal emulation with the remote computer, give the FINISH command and reconnect:

```
MS-Kermit>finish                    (Terminate mainframe session)
MS-Kermit>connect                   (Go back to remote computer)
Kermit-CMS>exit                     (Exit from remote Kermit)
.                                   (Back at mainframe prompt)
```

Now you can do further work on the mainframe. Remember to log out when you are finished.

MS-DOS Kermit is able to take advantage of most of the functions offered by the popular Kermit servers. You will find this mode of operation most convenient if you need to transfer collections of files in both directions while checking your directory on the host, deleting files there, and the like. If you transfer only the occasional file, then the SEND/RECEIVE style of operation might be more convenient.

Commands for Transferring Files with Kermit Servers

These are the commands that MS-DOS Kermit sends to remote Kermit servers. All of them can be interrupted by pressing the X, Z, C, Q, or E keys in the same way as when using MS-DOS Kermit to SEND or RECEIVE.

SEND

The SEND command is the same as always. It tells MS-DOS Kermit to deliver a file or file group to the server. You can also include a different name to send a file under:

```
MS-Kermit>send boring.txt exciting.txt
```

GET

The GET command sends a special command to the server, telling it the name of the file you want it to send to your PC:

```
MS-Kermit>get mail.txt
```

When communicating with a Kermit server, you must use the GET command rather than the RECEIVE command. If you use RECEIVE with a server, the server will not know what to do, and MS-DOS Kermit will wait a long time for a file that is not going to come. If you get into this situation, press the C key to get the Kermit prompt back immediately. You can specify a new name for the file when it arrives at your PC by typing GET alone on a line. You will be prompted separately for the two names:

```
MS-Kermit>get
 Remote Source File: profile exec
 Local Destination File: profile.xec
```

If you change your mind at one of the filename prompts and decide you don't want to issue the GET command after all, type *Ctrl-C* (hold down the Ctrl key and press the letter C) to get the MS-Kermit> prompt back.

BYE

The BYE command tells the remote Kermit server to terminate itself and your entire session on the host. If you give the BYE command, you don't have to (and probably cannot) connect back to the host and log out. The BYE command also causes MS-DOS Kermit to exit and return to DOS.

LOGOUT

The LOGOUT command is exactly like BYE, but it does not exit from MS-DOS Kermit.

FINISH

The FINISH command tells the server to get out of server mode and return to its interactive Kermit prompt so that you can connect back to the host and do more work.

The Server's Remote File Services

But wait, there's more! Besides sending and receiving files, the remote Kermit server offers a selection of file management and host access functions, all accessible from your PC's `MS-Kermit>` prompt. The key to these services is the REMOTE command. When you give a REMOTE command, it is sent to the remote Kermit server and the results are sent back to your screen. If you want the results to go to a file on your PC instead, use the DOS output redirection symbol, `>`, followed by a DOS file or device name:

```
MS-Kermit>remote directory x*.* > \lucy\dir.txt
```

If your REMOTE command needs to contain a `>` character, include a redirection phrase like `> CON` at the end of the command (CON is the device name of your PC's screen):

```
MS-Kermit>remote host sort < xx > yy > con
```

The final `> con` is used by MS-DOS Kermit for redirection; the other items are sent to the remote Kermit server.

If a REMOTE command gives you an error message like "Unable to open CON" this means that DOS has passed its limit on open files. You can increase the limit by including a line like `FILES=40` in your `CONFIG.SYS` file. Here are MS-DOS Kermit's REMOTE commands:

REMOTE CD

Changes the working directory on the host. You should specify the name of the directory to change to. If you give this command without including a directory name, the server will change to your default directory on the remote host:

```
MS-Kermit>rem cd /usr/jrd
/usr/jrd
MS-Kermit>rem cd
/usr/fdc
MS-Kermit>
```

The new directory will be the directory used for all file operations on the host unless you specify otherwise. If a password is required, add the password on the same line:

```
MS-Kermit>rem cd ps:[sy.fdc] secret
```

REMOTE DELETE

Asks the server to delete the specified files on the remote host computer:

```
MS-Kermit>remo del *.obj
MS-Kermit>remo del margaret/*.*
MS-Kermit>
```

You can delete only those remote files for which you have write or delete permission.

REMOTE DIRECTORY

Lists files on the remote computer. If you don't include a directory or file specification, all files in the current directory will be listed. If you do include a directory or file specification, the files that match are listed:

```
MS-Kermit>remo dir o*.*
total 2
-rw-rw---- spg  224 Feb  8 15:02 oofa
-rw-rw---- fdc  115 Nov 10 15:02 oofa.c
-rw-rw---- spg 1287 Jun 11 15:02 oofa.sh
MS-Kermit>
```

REMOTE HELP

Asks the server to list the services it provides. Kermit program services may vary. Here, for example, is the response from the VAX/VMS C-Kermit server:

```
MS-Kermit>remote help
C-Kermit Server REMOTE Commands:

GET files  REMOTE CD [dir]     REMOTE DIRECTORY [files]
SEND files REMOTE SPACE [dir]  REMOTE HOST command
MAIL files REMOTE DELETE files REMOTE WHO [user]
BYE        REMOTE PRINT files  REMOTE TYPE files
FINISH     REMOTE HELP
```

This tells you which commands the server can execute for you, for example:

```
MS-Kermit>remote space [catherine]
```

If you type an MS-DOS Kermit server-related command that is not on the server's list, you will get an error message:

```
MS-Kermit>remote message Guess who?
Error: Unimplemented server function
MS-Kermit>
```

The REMOTE commands (except for REMOTE HOST) are generic in that they are the same (if the program supports them) no matter which Kermit server you are connected to. For example, Kermit programs on the IBM mainframe, UNIX, VAX/VMS, and PDP-11 all use the command REMOTE TYPE to type a file, REMOTE DIRECTORY to list filenames, and

so on, even though each of these host computers might use different commands for the same requests if you were talking to them directly.

REMOTE HOST

The REMOTE HOST command, however, is specific to each kind of host. With this command, you tell MS-DOS Kermit to ask the host Kermit server to ask the host operating system to execute the given command, written in the host system's own command language. For example, here is how you would use the REMOTE HOST command to ask the Kermit server to rename a file on various kinds of hosts (MS-DOS Kermit currently does not have a REMOTE RENAME command):

```
MS-Kermit>rem host mv oofa.new oofa.old          (UNIX)
MS-Kermit>rem host rename oofa.new oofa.old      (VAX/VMS)
MS-Kermit>rem host rename oofa new a oofa old a  (VM/CMS)
```

The command that you ask the remote host to execute *cannot be an interactive command* that requires a response from the user.

REMOTE KERMIT

Sends a command to the remote Kermit server in its own command language, for example:

```
MS-Kermit>remote kermit set block 2
```

Very few Kermit servers support REMOTE KERMIT requests (IBM mainframe Kermit does). REMOTE SET (see below) can usually be used instead.

REMOTE LOGIN

Logs in to a remote Kermit server that has been set up to require a username and password:

```
MS-Kermit>rem login leslie anything
```

Here, "leslie" is the username, and "anything" is the password. You can use braces to group multiword items together:

```
MS-Kermit>rem login {first last} always
```

If the username and password are omitted from the command line, you will be prompted for them. If the password is given on the REMOTE LOGIN command line, it echoes, but if you're prompted for it, it doesn't:

```
MS-Kermit>rem login ken secret
MS-Kermit>rem login
 Username: ken
 Password: ______
 Account:
```

At the `Account:` prompt, type your account, if one is required, or just press the Enter key. Note: as yet, very few Kermit servers, other than MS-DOS Kermit itself, support REMOTE LOGIN.

REMOTE MESSAGE

Sends a one-line message for display by the remote Kermit server. This command is useful when the remote Kermit server is a PC with a person looking at the screen, for example:

```
MS-Kermit>rem mess OK, I'm finished...
MS-Kermit>rem mess You can sleep now!
```

REMOTE PRINT

Sends the specified file to the remote Kermit and asks the remote Kermit to print it on a remote printer with the specified options, if any. Options come after the filename and are expressed in the syntax of the remote computer's print command. Examples:

```
MS-Kermit>rem print oofa.txt
MS-Kermit>rem print oofa.txt /copies=5
MS-Kermit>rem print oofa.txt -Plaserwriter
```

REMOTE SET

Changes one of the remote Kermit server's settings, for example:

```
MS-Kermit>remote set ? One of the following:
 Attributes  File  Incomplete  Block-check  Receive
 Retry  Server  Transfer  Window-slots
MS-Kermit>remote set file ? One of the following:
 Type  Names  Collision  Incomplete
MS-Kermit>remote set file type ?
 Text  Binary
MS-Kermit>remote set file type binary
```

REMOTE SPACE

Displays the amount of disk space available on the remote host so that you can estimate if it has enough room for the files you plan to send. You can specify a device or directory name, or you can leave it out to get a report about the current device and directory:

```
MS-Kermit>remote space
MS-Kermit>rem spa $disk1:[donna]
```

REMOTE TYPE

Displays a remote file on your PC screen:

```
MS-Kermit>remote type max.txt
```

REMOTE WHO

Lists the users who are logged on the remote computer or gives you information about the specified user:

```
MS-Kermit>remote who            (All users)
MS-Kermit>remote who anne       (Specific user)
```

The results depend on the host operating system.

The MAIL Command

MS-DOS Kermit's MAIL command lets you send electronic mail to other users. This command can be used in conjunction with either a Kermit server or a Kermit program that has been given the RECEIVE command. MAIL only works if the remote Kermit supports this feature.

The MAIL command sends the specified file to the remote Kermit program, just like the SEND command does. But a "disposition attribute" is also included that tells the receiving Kermit to send the file as electronic mail to the address you specify after the filename, which can be a local user:

```
MS-Kermit>mail oofa.txt frank
```

or a network address:

```
MS-Kermit>mail question.msg info-kermit@watsun.cc.columbia.edu
MS-Kermit>mail answer.msg vax2::newuser
```

The remote Kermit program receives the file into its current directory and then tells the system mailer to send it as electronic mail to the address you specified. Once the file has successfully entered the "postal system," the copy received by the remote Kermit is deleted. Kermit programs that can be MAILed to include C-Kermit 5A for UNIX and VAX/VMS, and Kermit-370 4.2 for IBM mainframes.

Chapter 11

Making Your PC the Remote Computer

MS-DOS Kermit is normally used to *initiate* a connection to a remote host or service. But a PC with Kermit can also act as a remote host itself and receive calls from other computers. There are two ways to do this.

Method 1: Server Mode

The safer and more secure method of remote operation is MS-DOS Kermit's server mode. To use the server option, start MS-DOS Kermit and set the desired speed and other parameters as you've done before. Then type the SERVER command. If your PC is connected to a modem, you must also put the modem in *answer mode*. On a Hayes or Hayes-compatible modem, the command is ATS0=1 (that is a zero, not the letter O):

```
C>kermit                        (Run Kermit)
MS-Kermit>set speed 2400        (Set dialup speed)
MS-Kermit>connect               (Connect to the modem)
ATS0=1                          (Put modem in answer mode)
Alt-X                           (Escape back)
MS-Kermit>server                (Enter server mode)
```

The SERVER command can also accept an operand that tells how many seconds to remain in server mode before returning to the `MS-Kermit>` prompt:

```
MS-Kermit>server 3600           (Serve for 1 hour)
```

or until what time to run in server mode:

```
MS-Kermit>server 22:00:00       (Serve till 10:00 P.M.)
```

This way, you can tell your friend "Call my PC between 9:00 and 10:00 P.M. and get the OOFA file." But if your friend forgets to call, your PC won't be sitting in server mode indefinitely. In fact, using this mechanism you can schedule your PC to automatically run different jobs at different times of day. To find out how to start the server (or any other Kermit operation) at a specified time, read about the PAUSE command in Chapter 14.

Remote Commands

The MS-DOS Kermit server supports the following commands from the client Kermit:

SEND	REMOTE CD	REMOTE LOGIN
GET	REMOTE DELETE	REMOTE MESSAGE
FINISH	REMOTE DIR	REMOTE SEND
BYE	REMOTE HELP	REMOTE SET
LOGOUT	REMOTE HOST	REMOTE SPACE
	REMOTE KERMIT SET	REMOTE TYPE

These commands behave in the normal way; for example, to get a directory listing from the remote server's disk:

```
MS-Kermit>rem dir oofa          (Ask for list of remote files)
 Volume in drive C is BENICE
 Directory of  C:\TAKA

OOFA     C       6583   3-31-88  10:50a
OOFA     EXE     8335   3-31-88  10:50a
OOFA     HLP      655   7-30-89   5:51p
        3 File(s)   1445888 bytes free
MS-Kermit>
```

The REMOTE HOST command lets you run any DOS command on the server PC, displaying the output on your screen. The following example gets a listing of all the subdirectories of the current remote directory:

```
MS-Kermit>rem host dir | find "<DIR>"
SPSG         <DIR>      2-08-89   4:03p
SRSKM        <DIR>     11-10-89  11:32a
MS-Kermit>
```

Be careful not to invoke *interactive* remote PC commands like the line editor EDLIN. There is no way to carry on a dialog with the remote application through a Kermit server. If you need to do that, redirect the console with the DOS CTTY command as described in the section on the second method of remote operation, "Redirecting the DOS Session."

Security Features

The MS-DOS Kermit server lets users change directories, read files, create files, and even delete files, as well as run any DOS command. If you don't want to expose your PC to that kind of risk, you can use the DISABLE command to remove selected server functions:

DISABLE CD
(or CWD) Entirely disables changing of directories.

DISABLE DELETE
Confines deletion of files to the current directory.

DISABLE DIRECTORY
Confines production of directory listings to the current directory.

DISABLE FINISH
Entirely disables shutting down the server (applies also to BYE).

DISABLE GET
Confines getting files from the server to the current directory.

DISABLE HOST
Entirely disables execution of all REMOTE HOST (DOS) commands.

DISABLE KERMIT
Disallows use of the REMOTE KERMIT command.

DISABLE LOGIN
The REMOTE LOGIN command is not required.

DISABLE PRINT
Disables REMOTE PRINT commands from the other Kermit.

DISABLE SEND
Forces files sent to the server into the current directory. There is no way to completely disable the reception of files.

DISABLE SPACE
Disables asking the server for a disk space report with the REMOTE SPACE command.

DISABLE TYPE
Confines the REMOTE TYPE command to the files in the current directory.

DISABLE ALL
All the above.

A disabled function can be turned back on with the ENABLE command, for example:

```
MS-Kermit>enable login
```

You can obtain an extra level of security by setting a username and a password for server access:

```
MS-Kermit>set server login name password
MS-Kermit>server
```

This requires the user, after establishing the dialup connection, to give the command:

```
MS-Kermit>remote login name password
```

If the server has been set up to require a login, and you send any other command to it before logging in, it will respond with a message to the effect that login is required:

```
The server PC is set with the following security:
MS-Kermit>set prompt MS-Server>
MS-Server>disable all
MS-Server>enable login
MS-Server>set server login linda secret
MS-Server>server

Someone tries to access this PC without access codes:
MS-Kermit>get oofa.txt
REMOTE LOGIN is required

Someone tries to access this PC with wrong access codes:
MS-Kermit>remote login raynette alf
Invalid login information

Someone accesses the PC with correct access codes:
MS-Kermit>remote login linda secret
MS-Kermit>
```

Disengaging from the Server

The user can terminate the server session on your PC by using the BYE or LOGOUT commands described in Chapter 10. If you want someone else to be able to call your PC server, start it this way:

```
MS-Kermit>disable finish
MS-Kermit>server
```

Then, if a user gives a BYE or LOGOUT command, the server will remain active and waiting for the next user to call.

Method 2: Redirecting the DOS Session

This second method of obtaining remote access to the PC is not recommended, but since it is sometimes used, it should be discussed.

The DOS command CTTY can be used to redirect a DOS session from the normal console (PC keyboard and screen) to a communication port:

```
C>ctty com1
```

After this command is issued, the DOS command prompt moves to COM1, and DOS accepts commands only from COM1, not from the keyboard. Now other computers or terminals can access your PC by connecting to (or dialing up) your COM1 port. If you have a modem attached to your PC so that people can dial up, you should put it in answer mode.

When accessing DOS through the communication port, you can use only "well-behaved" DOS-level character-mode applications like EDLIN. Any application that puts graphics on the screen or that requires Alt or function (F) keys cannot be used. If you start such an application, you'll be locked out. Even well-behaved applications can lock you out if they get an error because the familiar DOS error message:

```
Abort, Retry, Fail?
```

appears on the real screen and expects a reply on the real keyboard.

If you understand the limitations and risks of using DOS in this manner, you can make effective use of it within this framework. Let's say you want to be able to access your office PC from your home PC and that you have Hayes modems for each PC. Here is the last thing you would do before leaving work:

```
C>kermit                        (Run Kermit at work)
MS-Kermit>set speed 2400        (Set dialup speed)
MS-Kermit>connect               (Connect to the modem)
AT                              (Make sure modem is working)
OK                              (It is)
ats0=1                          (Put it in answer mode)
Alt-X                           (Escape back to Kermit)
MS-Kermit>exit                  (Exit from Kermit)
C>ctty com1                     (Now move DOS to COM1)
```

When you get home (after eating dinner), you can dial up your office PC:

```
A>kermit                        (Run Kermit at home)
MS-Kermit>set speed 2400        (Set dialing speed)
MS-Kermit>connect               (Connect to your home modem)
AT                              (Make sure it's working)
OK                              (It is)
atdt7654321                     (Call your office number)
CONNECT 2400                    (Hayes says connection is made)
                                (Press Enter key)
C>                              (DOS prompt from office PC)
```

When you are dialing a remote PC from another PC, both Kermit programs will have the same prompt, which can be confusing. You can use Kermit's SET PROMPT command to give distinct prompts to the two programs so that you'll always know which one you're talking to:

```
MS-Kermit>set prompt Work>      (This one is at the office)
Work>                           (Here's the new prompt)
Alt-X                           (Escape back to home PC)
MS-Kermit>set prompt Home>      (This one is at home)
Home>                           (New prompt on home PC)
Home>connect                    (Connect to the remote PC)
Work>receive                    (I'd better send OOFA.TXT)
Alt-X                           (Escape back to home)
Home>send oofa.txt              (No PC is complete without it)
```

To restore your work PC to normal, do this:

```
Work>exit                       (Exit from remote PC Kermit)
C>ctty con                      (Put its console back to normal)
Alt-X                           (Escape back to home PC)
Home>hangup                     (Hang up the phone)
Home>exit                       (And exit back to DOS)
A>
```

CTTY CON directs DOS back to the real keyboard and screen on your PC at work. You cannot do this from the work PC's keyboard except by rebooting the PC with *Ctrl-Alt-Del* (press the Ctrl, Alt, and Del keys simultaneously).

If you have redirected DOS to a port other than COM1, you must also tell MS-DOS to SET PORT to the same port before attempting to transfer files, or file transfer will not work.

Aside from the operational risks of remote DOS operation, there is a security risk. Anybody who knows your phone number can call your PC and wreak havoc with your files. DOS has no built-in security features like user IDs or passwords to regulate access. This is why it's important to put your work PC back to normal when you're finished using it. That way, if anyone else dials it up, they won't be able to communicate with it at all.

Chapter 12

Transferring Files without the Kermit Protocol

Now that you know how to transfer files with another computer that has a Kermit program, it's time to consider the unthinkable: What if the other computer *doesn't have Kermit?*

You have several courses of action:

- Badger the computer's system manager mercilessly until Kermit is installed.
- Get Kermit and install it yourself.
- If a Kermit program doesn't exist for a particular computer (there are still a few), write one!
- Use MS-DOS Kermit to transfer files without error correction.

This chapter discusses the last alternative. You already know how to download a file without error correction. If you don't believe this, review the section on session logging in Chapter 8. How can you use session logging to download a file? Simple: Just display the file on the remote computer while logging the session. For uploading files to computers that don't have Kermit, there is a special command, TRANSMIT.

But (you may ask) if that's all there is to it, what do I need the Kermit protocol for? Here are just a few reasons: You can't transfer binary files this way, you can't transfer more than one file at a time, transmission errors and data loss can't be detected, filenames and

other characteristics are not transmitted, and the whole process is cumbersome and unreliable. But it *is* better than nothing.

For both uploading and downloading, you must first get connected and logged in, setting all the appropriate communication parameters, with particular attention to duplex, flow control, and handshake.

Downloading a Host File to the PC

The trick here is to type the host command to display the desired file *up to but not including the Enter key*, which actually starts the command. Then escape back to MS-DOS Kermit, start the session log, connect back to the host, and press the Enter key to start the command. When you are done, escape back again and close the session log.

Let's look at an example. Here we log in to a UNIX system, with which Kermit can use its default parameters and where the command to display a file is "cat" (yes, cat).

```
C>kermit                          (Start Kermit on the PC)

MS-Kermit>connect                 (Begin terminal emulation)
                                  (Press the Enter key)
login: marilyn                    (Log in)
Password: ________                (Type your password)

$ cat resume.txt                  (Type the command to display the)
                                  (file, but don't press Enter yet!)

Alt-X                             (Hold down the Alt key and press X)
                                  (to escape back to MS-DOS Kermit)

MS-Kermit>log sess resume.hlp     (Start the session log)
MS-Kermit>connect                 (Go back to the host)
                                  (Now press the Enter key)
```

As the host file is displayed on your screen, it is also recorded in the PC file with the name you specified in the LOG SESSION command.

This process does not terminate by itself. You have to watch. When you see the end of the file and the host computer prompt appears again, escape back to MS-DOS Kermit and close the session log:

```
(many lines are displayed...education, etc.)
  References will be provided upon request.
$                                (Host computer prompt appears)
Alt-X                            (Hold down the Alt key and press X)
MS-Kermit>close session          (Close the session log)
```

A copy of the file is now on your DOS disk. It is almost guaranteed to be somewhat different from the original copy, if only because it has the host prompt at the end. You may also find terminal messages mixed in, gaps, or unexpected characters caused by interference. If you want the file to be clean and pure, you have to go over it carefully with a text editor on your PC.

Uploading a PC File to the Host

To send a file to a host that doesn't have Kermit, you can use a combination of MS-DOS Kermit's TRANSMIT command and the host's file-creation mechanism.

Kermit's TRANSMIT command works only with text files. It sends the file as if you were typing it, a line at a time. Full-duplex flow control or half-duplex handshake is used to make sure that Kermit does not send the text faster than the host can store it, but there is no provision at all for error detection or correction.

On the host end, you should do whatever you normally do to create a new file whose text is entered from the terminal. In many cases, this means starting a line-oriented text editor and putting it in text insertion mode. Some hosts have commands that let you type text directly from the keyboard into a file. Table 12-1 shows some examples (for creating a file called `oofa.txt`).

Table 12-1 File Creation Commands

System	*To Begin*	*To End*
UNIX	`cat > oofa.txt`	*`Ctrl-D`*
VAX/VMS	`create oofa.txt`	*`Ctrl-Z`*
MS-DOS	`copy con oofa.txt`	*`Ctrl-Z`*, *`Enter`*
Most Others	(Use a text editor)	

Let's try this with a VAX/VMS system:

```
C>kermit                          (Start Kermit on the PC)

MS-Kermit>connect                 (Begin terminal emulation)
                                  (Press the Enter key)
Username: mary                    (Log in)
Password: _________               (Type your password)

$ create oofa.txt                 (Copy terminal to file)

Alt-X                             (Hold down the Alt key and press X)
MS-Kermit>transmit oofa.txt       (Send the file)
MS-Kermit>connect                 (Go back to the host)
Ctrl-Z                            (Hold down the Ctrl key and)
                                  (press Z to close the file)
Exit                              (VMS confirms the file is closed)
$ dir /size/date oofa.txt         (Is it really there?)

Directory $DISK1:[MARY]

OOFA.TXT;1   2  8-FEB-1991 11:32:29.75

Total of 1 file, 2 blocks.

$ logout                          (Remember to log out)
Alt-X                             (Hold down the Alt key and press X)
MS-Kermit>                        (Back at MS-Kermit> prompt)
```

The file (or pieces of it!) is now on your VMS disk. If you care about its integrity, use a text editor like EDT on the VMS system to inspect it and make any necessary repairs.

Here is a more complicated example in which we upload a text file to an IBM mainframe through a direct (nondialup) linemode connection. The operating system is VM/CMS, which does not include a command like "cat" or COPY to copy keystrokes to a file. In this case, a text editor, XEDIT, will be used.

Unfortunately, there is a small hitch here. XEDIT leaves text insertion mode if it receives a blank line. So if the file to be uploaded contains empty lines, Kermit must be told to insert a nonblank character into each blank line. The command for this is:

SET TRANSMIT FILL-EMPTY-LINE NONE *or* **SPACE** *or character*

This tells Kermit what to do when transmitting an empty line. The default action, NONE, means just send the empty line. SPACE means send a space character (this won't work with XEDIT, but it might work with other editors). Otherwise, you can include a nonblank character like X:

```
MS-Kermit>set transm fill X
```

Let’s use this feature to transmit a familiar file to XEDIT on the IBM mainframe:

```
C>kermit                          (Start Kermit on the PC)
MS-Kermit>set speed 9600          (Set the speed)
MS-Kermit>set duplex half         (Connection is half duplex)
MS-Kermit>set flow none           (No Xon/Xoff flow control)
MS-Kermit>set handshake xon       (Must use handshake)
MS-Kermit>set parity mark         (Mainframe uses MARK parity)
MS-Kermit>connect                 (Begin terminal emulation)

VIRTUAL MACHINE/SYSTEM PRODUCT--CUVMB --PRESS BREAK KEY

Alt-B                             (Hold down the Alt key and press B)
                                  (to send a BREAK signal)
!
.login sari                       (Log in)

Enter password: XXXXXXXX          (Half duplex; password echos)

LOGON AT 15:28:14 EDT THURSDAY 02/08/91
VM/SP REL 5 04/19/88 19:39
.
CMS
.xedit oofa txt                   (Start the editor)
.i                                (Put it in text input mode)
DMSXMD573I Input mode:
Alt-X                             (Hold down the Alt key and press)
                                  (the X key to return to PC Kermit)

MS-Kermit>set transm fill X       (Fill blank lines with X)
MS-Kermit>transmit oofa.txt       (Send the file)
MS-Kermit>connect                 (Return to the mainframe)
                                  (Press the Enter key)
                                  (to leave text input mode)
DMSXMD587I XEDIT:
.save                             (Tell XEDIT to save file)
.qq                               (Quit from XEDIT)
Ready; T=0.02/0.06 15:29:11
.lf oofa                          (Make sure file is there)
OOFA     TXT                      (It is)
Ready; T=0.01/0.01 15:29:16
.type oofa txt                    (Take a look at it)

This is the first line of the file oofa txt.
X (This was a blank line)
...
And this is the last line.

Ready; T=0.01/0.01 15:29:20
.logout                           (Log out from the mainframe)
CONNECT= 00:01:11 VIRTCPU= 000:00.12 TOTCPU= 000:00.32
LOGOFF AT 15:29:25 EDT THURSDAY 02/08/91
Alt-X                             (Hold down Alt and press X)
MS-Kermit>
```

Chapter 13

International Character Sets

The need for computers to communicate with each other—to share information—has become universal. But until recently, most computer hardware and software vendors provided their customers only with products based on the English language and with character sets capable of representing only English text. These character sets contained the letters A–Z but none of the special characters required by other languages like French, German, Italian, Norwegian, Hebrew, Arabic, Greek, Russian, Chinese, or Japanese.

There are now computers capable of representing these special characters, but each may do so in a different way. For example, an importer of foods might create an order for pâté on an IBM PC, then transfer the order using Kermit (or any other file transfer protocol) to a supplier's Macintosh, where the word appears as "pÉtä" on the screen, which could easily result in a shipment of pita bread rather than goose liver.

An extension to the Kermit protocol handles this problem by specifying a common *transfer syntax* consisting of a small number of well-defined standard international character codes. During file transfer, each Kermit program converts between its own computer's character sets and the standard ones and needs no knowledge of any other computer's character sets.

MS-DOS Kermit 3.0 was the first Kermit program to follow the new protocol. MS-DOS Kermit's international character support includes not only file transfer but also terminal emulation.

IBM PC Character Sets

Characters are represented in the computer as sequences of bits, usually 7 or 8 bits per character. When such a bit sequence is transmitted to a display device like a terminal or printer, it is interpreted as a number that tells the position in the device's character generator. For example, the bit pattern 01000001 is equivalent to the decimal number 65, which might denote the letter A. When a video terminal receives this code, it retrieves the 65th element from its character table, which hopefully contains the dot pattern that looks like an A, and displays it on the screen.

When codes (bit sequences) are assigned to an alphabet (including letters, punctuation, and digits), the result is called a "character set." If every computer used a different character set, no two computers could communicate. For example, an A on one computer might be interpreted as a B on another, and that wouldn't be very useful. For this reason, standard character sets like ASCII[12] were developed (the ASCII character set is listed in Table I-5). Entire books are dedicated to the topic of character set design.[13] ASCII has 128 characters, including the Roman letters A through Z in upper and lower case, but not including accented letters or letters in other alphabets. ASCII cannot represent all the characters in every language, so other character sets are also required. And depending on your needs, you might have to use more than one of them.

IBM's character sets for the PC are called *code pages*. They are listed at the back of your DOS manual and in Table I-7 on page 277 of this book. Each of these character sets is a "superset" of ASCII that includes 128 additional printable characters, plus special graphic symbols like smiley faces, suits of cards (♣ ♦ ♥ ♠), musical notes, and arrows where the control characters usually reside (in true ASCII, the control characters do not have associated graphic symbols).

But the placement of the extra 128 characters differs from code page to code page. The original PC code page, CP437, includes about 56 "national characters" that are not defined in ASCII, such as u-diaeresis (ü), a-acute (á), c-cedilla (ç), and o-circumflex (ô); sufficient, according to the DOS manual, to support five different languages.

To capture markets in lands where the Five Languages are not spoken, IBM began to produce PCs with code pages for additional languages: a Portuguese code page, a Norwegian code page, and so forth. Eventually IBM added the multilingual (many-tongued) code page, CP850, which claims to support eleven languages and to make the earlier code pages obsolete.

---

[12]ANSI X3.4-1986, American National Standard Code for Information Interchange

[13]See, for example, *Coded Character Sets, History and Development* by C.E. Mackenzie, Addison-Wesley (1980) for a history of the EBCDIC and ASCII sets.

Accented letters, Greek letters, and so forth are referred to as *special characters*, even though there is nothing special about them to those who use them every day in their written languages. They are special only because they are not part of ASCII and because each computer manufacturer seems to have a different way of entering and representing them.

First, let's find out how to enter international text on a PC—how to type it at the keyboard and how it is displayed on the screen. The IBM PC-PS/2 family has two ways to do this, the new way and the old way. The new way is more flexible and convenient, but it can't be done on all PCs.

Code Page and Keyboard Switching

This is the new way. The features described in this section are available only on PCs or PS/2s with DOS 3.30 or later that have an Enhanced Graphics Adapter (EGA) or EGA-compatible or higher (VGA, XGA) video adapter. Skip this section if this does not apply to you.

Old code pages never die. They don't even fade away. CP850 was able to add new western European national characters only by removing the Greek and technical characters from the other code pages. But since many users have applications that depend on the characters that were sacrificed, DOS 3.30 was equipped with a new feature called *code page switching*. This lets files created with different code pages coexist on the same PC. To work with files created with a given code page, you must make that your *active* code page, using the DOS CHCP command, for example:

```
C>chcp 865
```

You can find out what DOS thinks your current code page is by typing `chcp` by itself:

```
C>chcp
Active code page:  865
```

Warning: Non-IBM PCs might not support the CHCP command, and might not support code page switching at all. Even when code page switching is supported, DOS might not always report the current code page correctly.

On most PCs, there are no keys labeled with special characters. How do you type a u-diaeresis or an A-ring when there are no keys for these characters? PCs may be equipped with "national keyboards," which do indeed have keys for the special characters of a particular language. But the problem remains. How do you type, say, Italian characters on a Norwegian keyboard? In DOS 3.30 or later, you may use the DOS KEYB command to reload your keyboard with the characters of a given national keyboard. Of course, this doesn't change the labels on your keycaps, so you have to refer to the appropriate keyboard template at the back of your DOS manual (Appendix E for IBM PC DOS 3.30) to see which key to press for which character.

You must load the code-page switching console driver, DISPLAY.SYS, before you can change code pages and keyboards. Add a line like this to your CONFIG.SYS file:

```
DEVICE=C:\DISPLAY.SYS CON:=(EGA,437,(4,2))
```

In this example, EGA is the display adapter type (use EGA even for VGA or XGA), 437 is the PC's built-in hardware code page (this can vary), 4 is the number of additional code pages you want to be able to use (8K is reserved for each code page), and 2 is the number of subfonts per code page, normally 2 (use 1 for the IBM PC Convertible).

You also need entries like the following in your AUTOEXEC.BAT file. This example assumes the relevant files are stored in the top-level directory of your C disk:

```
REM ... Keyboard and console code page
NLSFUNC C:\COUNTRY.SYS
MODE CON: CP PREPARE=((850,,437) C:\EGA.CPI)
MODE CON: CP PREPARE=((,866,) C:\XTRA.CPI)
MODE CON: CP SELECT=850
KEYB US,850,C:\KEYBOARD.SYS
```

These commands set your PC up with the United States keyboard layout and the multilingual code page on an EGA system. The line referring to CP866 illustrates how to prepare a custom code page (in this case, Microsoft's Cyrillic code page 866) from a (hypothetical) separate Code Page Information (CPI) file, XTRA.CPI. After adding these commands to your CONFIG.SYS and AUTOEXEC.BAT files, you must restart your PC to make them take effect.

Now if you need characters from CP437 (like the Greek letters), you can switch like this:

```
C>chcp 437
```

and if you need, say, Icelandic letters, you can switch back to CP850 like this:

```
C>chcp 850
```

You can switch among code pages as often as you like. A code page change affects your entire screen (and Kermit's rollback buffer); you can't mix characters from different code pages on the same screen.

You can change your keyboard at any time, too:

```
C>keyb it,850                   (Italian keyboard, CP850)
C>keyb                          (Check it)
Current keyboard code: IT  code page: 850
Current CON code page: 850
```

The first operand of the KEYB command is the country code, IT for Italy, NO for Norway, and so on. Your DOS manual should contain a table of the permissible country codes and which code pages may be used with them.

Most national keyboard configurations also allow special characters to be entered using two-character sequences. For example, the Norwegian configuration lets you type a wide variety of accented vowels that don't actually appear on the Norwegian keyboard by pressing the accent key and then the desired letter. For example, to produce e-grave (è), first press the grave key (nothing happens) and then press e (è appears). IBM calls these key sequences dead-key combinations. They are listed in the keyboard section of the DOS manual. Dead keys are not supported for U.S., U.K., or Italian keyboards.

Special Character Entry with Alt-Key Combinations

This is the old way of entering special characters, and it is available in any version of DOS or any PC model. On PCs without code page or keyboard switching, it is also the only way to enter special characters. To enter a special character, hold down the Alt key, press three digits on the numeric keypad, then let go of the Alt key. The three digits are the *decimal* representation of the PC character in the current code page. For example, if you hold down the Alt key and press the numbers 1 2 8 consecutively using most code pages, C-cedilla will be the resulting character on your screen. Unfortunately, the DOS manual code-page tables do not list the decimal values, only the *hexadecimal* values, which makes this method hard to use. Use Table I-7 in the back of this book instead, which lists the decimal codes as well as the hexadecimal codes for the special characters in five code pages.

Terminal Emulation

MS-DOS Kermit is fully capable of displaying and transmitting special characters during terminal emulation, provided it has the characters in its current code page and you have told Kermit to emulate the VT220 or VT320 terminal. Host computers such as UNIX, VAX/VMS, and IBM mainframes use different codes from the PC to represent national characters. During terminal emulation, MS-DOS Kermit must translate the characters sent by the host computer into the PC's current code page for display on the screen, and it must translate the characters you type on the PC keyboard from the PC code page into the host computer's character set. To perform the correct translations, MS-DOS Kermit must be told which character set the host computer is using. The command is:

SET TERMINAL CHARACTER-SET *name*

You can pick any *name* from Table 13-1, for example:

```
MS-Kermit>set term char german
```

Table 13-1 shows the character sets supported by MS-DOS Kermit's VT220 and VT320 terminal emulation. These sets fall into two categories: 7-bit character sets (128 characters each) and 8-bit character sets (256 characters each), as shown in the *Bits* column.

Table 13-1 MS-DOS Kermit Terminal Character Sets

Name	*Bits*	*Characters*	*Description*
ASCII	7	128	ASCII
BRITISH	7	128	British (UK) NRC
DEC-SPECIAL	7	128	DEC Special Graphics (line and box drawing)
DUTCH	7	128	NRC for the Netherlands
FINNISH	7	128	Finnish NRC
FRENCH	7	128	French NRC
FR-CANADIAN	7	128	NRC for French Canada
GERMAN	7	128	German NRC
ITALIAN	7	128	Italian NRC
NORWEGIAN/DANISH	7	128	NRC for Norway and Denmark
PORTUGUESE	7	128	Portuguese NRC
SPANISH	7	128	Spanish NRC
SWEDISH	7	128	Swedish NRC
SWISS	7	128	NRC for Switzerland
DEC-MCS	8	256	DEC Multinational Character Set
LATIN1	8	256	ISO Latin Alphabet Number 1 (default)
TRANSPARENT	8	256	Current IBM PC code page

The 7-bit sets include ASCII and country-specific variations of ISO 646 (the European counterpart to ASCII) called National Replacement Character (NRC) sets. In an NRC set, certain nonalphabetic ASCII characters like `@`, `[`, `\`, `]`, `{`, `|`, `}`, and `~` are replaced by the characters needed for each language, as shown in Table 13-2 (a more complete listing is given in Table I-6). The NRCs are for use in 7-bit transmission environments, when parity prevents the use of the full 8 data bits.

The 8-bit terminal character sets include the international standard ISO Latin Alphabet Number 1 and the DEC Multinational Character Set (very similar to Latin Alphabet 1). In most international environments that use Roman characters and allow 8-bit character transmission (no parity), Latin Alphabet 1 is used. MS-DOS Kermit's default terminal character set for VT220 and VT320 emulation is LATIN1.

According to ISO Standard 8859-1, Latin Alphabet Number 1 supports at least the following languages: Danish, Dutch, English, Faeroese, Finnish, French, German, Icelandic,

Table 13-2 Selected National Replacement Character Sets

Code	*ASCII*		*Italian*		*Norwegian*		*French*		*German*	
64	@	at sign	§	paragraph	@	at sign	à	a-grave	§	paragraph
91	[	left bracket	°	degree	Æ	AE-digraph	°	degree	Ä	A-diaeresis
92	\	backslash	Ç	c-cedilla	Ø	O-slash	ç	c-cedilla	Ö	O-diaeresis
93	]	right bracket	é	e-acute	Å	A-ring	§	paragraph	Ü	U-diaeresis
96	‘	accent grave	ù	u-grave	‘	accent grave	‘	accent grave	‘	accent grave
123	{	left brace	à	a-grave	æ	ae-digraph	é	e-acute	ä	a-diaeresis
124	|	vertical bar	ò	o-grave	ø	o-slash	ù	u-grave	ö	o-diaeresis
125	}	right brace	è	e-grave	å	a-ring	è	e-grave	ü	u-diaeresis
126	~	tilde	ì	i-grave	~	tilde	¨	diaeresis	ß	eszet

Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish (to which we can add Latin itself for a total of fifteen supported languages).

Before you can use an 8-bit terminal character set, you should ensure that you have an 8-bit connection to the host computer, and then issue the MS-DOS Kermit commands SET PARITY NONE and SET TERMINAL BYTESIZE 8. Refer to Table 13-1 to determine which character sets are 7 bits and which are 8 bits.

You can also use an 8-bit character set in the 7-bit environment if the host computer supports a communications technique called shift-in/shift-out; see page 289 in Appendix II.

Customizing the Terminal Character Set

MS-DOS Kermit’s built-in terminal character sets are standard and should meet the needs of those who must display characters in any of the languages listed. But you can override Kermit’s translation of incoming characters on an individual basis using the SET TRANSLATION INPUT command. The format is:

SET TRANSLATION INPUT *input-character screen-character*

which causes Kermit to translate the specified incoming character to the specified screen character. You may issue as many of these commands as you like. This lets you adapt MS-DOS Kermit to virtually any host character set. For example, there is a second, “programmer’s,” NRC in use in Sweden. To modify Kermit’s built-in Swedish NRC to use the programming version, issue these commands:

```
MS-Kermit>set terminal character-set swedish
MS-Kermit>set transl in \144 \64  E-acute to atsign
MS-Kermit>set transl in \154 \94  U-diaeresis to circumflex
MS-Kermit>set transl in \129 \126 u-diaeresis to tilde
MS-Kermit>set transl in \130 \96  e-acute to accent grave
MS-Kermit>set transl in on        Enable translation
```

This is called a user-defined translation, and it takes precedence over the currently selected terminal character set for the affected characters only. Input translation is normally OFF. You have to turn it ON in order to use it. Turning it OFF again does not destroy the translation table you have created, but only disables its use. Once you have created the table you can SET TRANSLATION INPUT ON and OFF as needed.

Adding a New Terminal Character Set

Suppose your PC has a code page or character set that Kermit doesn't know about, or the remote host is using a character set unknown to Kermit. If the host's character codes agree with your PC's code page, just tell Kermit to SET TERMINAL CHARACTER-SET TRANSPARENT to see the correct characters on your screen.

If the host's character codes do not agree with your PC code page, use SET TRANSLATION INPUT commands to make the translations. For example, a Russian PC that uses code page 866 and accesses a host computer that uses the ISO 8859-5 Latin/Cyrillic character set (see Table I-8) would need one SET TRANSLATION INPUT command for each character to be translated:

```
set term byte 8           ; Use 8-bit characters
set term char transparent ; Use untranslated host values
set transl in \176 \128   ; Cyrillic A
set transl in \177 \129   ; Cyrillic Be
set transl in \178 \130   ; Cyrillic Ve
...
set transl input on       ; Turn on translation
```

If the host's character set has characters that are not in your PC code page at all, you can use SET TRANSLATION INPUT commands to achieve rough equivalences. In this example, the host uses Latin/Cyrillic but the PC uses CP437, which contains no Cyrillic characters. Here we tell Kermit to display all Roman letters in uppercase and all Cyrillic letters in lowercase Roman:

```
set term byte 8           ; Use 8-bit characters
set term char transparent ; Use untranslated host values
set transl in \176 \97    ; Cyrillic A  to Roman a
set transl in \208 \97    ; Cyrillic a  to Roman a
set transl in \177 \98    ; Cyrillic Be to Roman b
set transl in \209 \98    ; Cyrillic be to Roman b
set transl in \178 \118   ; Cyrillic Ve to Roman v
set transl in \210 \118   ; Cyrillic ve to Roman v
...
```

```
set transl in \97  \65    ; Roman a to Roman A
set transl in \98  \66    ; Roman b to Roman B
...
set transl input on       ; Turn on translation
```

The result is a notation called "Short KOI," commonly used in the Soviet Union for displaying Cyrillic text on ASCII devices.

The SET TRANSLATION INPUT command works differently depending on the current TERMINAL CHARACTER-SET. If it is TRANSPARENT, Kermit translates the *input-character* directly to the *screen-character*, as in the Cyrillic examples. If a non-transparent character set is active, as in the Swedish example, Kermit first translates the incoming character to the current PC code page before doing the translation that you asked for. So in this case, the *input-character* must be a PC code page value rather than the code sent by the host. If your host character set has graphic (printable) characters in the 128-159 range, you must SET TERMINAL CHARACTER-SET TRANSPARENT or else Kermit will treat these as 8-bit control characters.

Screen Writing Direction

Kermit's normal direction of screen display during connect mode is left to right. You can reverse this for languages like Hebrew and Arabic (assuming you have a PC with the appropriate code page) using the command SET TERMINAL DIRECTION RIGHT-TO-LEFT. This command will work for other languages (like English) too:

```
MS-Kermit>set term dir right
MS-Kermit>connect
                                                    gmc :emanresU
                                                        :drowssaP
                                              0.5 SMV/XAV ot emocleW
                                                              $
```

Leonardo da Vinci would have liked this command.

Keyboard Translations

Besides governing how incoming characters are translated for display on the screen, the SET TERMINAL CHARACTER-SET command also causes Kermit to translate any special characters that you type at the keyboard from the current PC code page into the host character set you have selected. Recall that you can type special characters using any of the methods described above: Alt-digit-digit-digit, dead key, specially labeled keys on national keyboards, or by loading a soft[14] national keyboard with the DOS KEYB command.

[14]Soft is computer slang for an effect accomplished by software.

Keyboard translation is in effect by default, but you can turn it off with the command SET TRANSLATION KEYBOARD OFF.

Just as you can override the translation of characters that arrive *from* the host with the SET TRANSLATION INPUT command, you can also override the translations of your keystrokes before they are sent *to* the host using the SET KEY command. For example:

```
MS-Kermit>set key \2334 \91
```

makes *Alt-a* send the German NRC code for a-diaeresis. You can use SHOW KEY to find out scan codes (or consult Table I-9). The NRC a-diaeresis value came from Table 13-2. This use of SET KEY requires you to have explicit knowledge of the host character set codes for special characters.

The SET KEY command is especially useful when using a code page that Kermit does not have built-in tables for, like CP855 or CP866 (Cyrillic), CP851 or CP869 (Greek), CP857 (Turkish), CP862 (Hebrew), CP864 (Arabic), CP868 (Urdu), or CP874 (Thai). For example, if your PC has CP866 and a Cyrillic keyboard and you want it to transmit ISO Latin/Cyrillic codes, you will need a SET KEY command for each lowercase letter and each uppercase letter, specifying the key scan code and translation value for each:

```
set key \128 \176   ; Uppercase letter A
set key \129 \177   ; Uppercase letter Be
set key \130 \178   ; Uppercase letter Ve
set key \131 \179   ; Uppercase letter Ghe
...
```

Taking Advantage of Kermit's Terminal Character Sets

If you are a European computer user, the benefits of Kermit's multinational terminal emulation are obvious. You have probably been using a "national" character set for years, and now Kermit gives you a more convenient way to do this on your PC—you can type your ñ key, the correct code is automatically sent to the host, and the host's echo is automatically displayed as ñ on your screen.

In the English-speaking world, the benefits are not so obvious. But consider this: Except within certain very restricted environments, we have never been able to:

- Spell our overseas customers' name and addresses correctly.
- Correspond with an overseas client in his or her own language (even if we knew it).
- Include foreign words and phrases in our documents.

Granted, this kind of thing was sometimes possible if we restricted ourselves to only one manufacturer's PCs and printers. Now we can use our PCs in conjunction with diverse

mainframes, minicomputers, dialup services, and with other kinds of PCs that use different character sets, and Kermit will do all the necessary translation.

When the Host Computer Has No Special Characters

You can use MS-DOS Kermit to teach an English-only host computer a new language. Here's one possible way to do it for traditional UNIX.[15] Simply SET TERMINAL CHARACTER-SET to the NRC set for the desired language, say, GERMAN. Enter the special German characters (Ä, Ö, Ü, ä, ö, ü, and β) using the German keyboard layout selected by the DOS command:

```
KEYB GR, 850
```

or by using the Alt-key combinations shown in Table I-7. This setup allows all UNIX host applications to display your German characters correctly, for example:

```
C>chcp 850
C>keyb gr,850
C>kermit
MS-Kermit>set term vt320
MS-Kermit>set term char german
MS-Kermit>connect
Grüße aus UNIX!
$ help
UNIX hilft dem, der sich selbst hilft!
$
```

This method is easy, but has the disadvantage of making brackets, braces, backslash, vertical bar, and tilde unavailable to you because these have been replaced by the German letters. If your host computer allows 8-bit connections (and you have one), and its applications (text editors, compilers, etc.) allow 8-bit data, just pick an 8-bit terminal character set like LATIN1 and use it (SET TERM CHAR LATIN1). The host doesn't need to know a thing about it, and you no longer have to sacrifice ASCII characters to get accented letters.

Printing International Characters

The international characters that appear on your screen during terminal emulation can also be printed correctly on a printer attached to your PC under certain conditions. Recall that transparent printing sends incoming characters direct to the printer without translation. If you want these characters printed correctly, your printer must support the host's character

[15]UNIX has always been a 7-bit ASCII environment, so 8-bit international character sets could not be used. The current trend is for each vendor to modify UNIX to be "8-bit clean," but it may take a long while for the results of this effort to become generally available. Try the UNIX command "stty pass8" to see if your UNIX supports 8-bit communication.

set, or your DOS print driver must do the translation from the host character set to the printer's character set.

The other printing methods (Print Screen, Autoprint, etc.) send host characters to the printer *after* Kermit has translated them into the current PC code page. In these cases, you need an IBM or other printer that supports your PC code page, or a print driver that translates appropriately. Recent IBM printer models support code page switching via CHCP (see your DOS or printer manual for details).

Terminal Emulation Summary

Here is a brief recapitulation of the steps necessary to send and receive international characters to and from the host during terminal emulation:

1. Find out what your current code page is:

   ```
   C>chcp
   Active code page: 437
   ```

2. Look up your current code page at the back of the DOS manual or in Table I-7 in the back of this book. Make sure it contains all the characters you need to use. If it does not, and if you are using DOS 3.30 or higher and have an EGA-compatible display board, install the code page you need by using a text editor (like EDLIN) to add the following lines to your `AUTOEXEC.BAT` file:

   ```
   NLSFUNC C:\COUNTRY.SYS
   MODE CON: CP PREPARE=((850,,437) C:\EGA.CPI)
   MODE CON: CP SELECT=850
   KEYB US,850,C:\KEYBOARD.SYS
   ```

   This example assumes that the DOS files `COUNTRY.SYS`, `KEYBOARD.SYS`, and `EGA.CPI` are in the top-level directory of your C disk. The MODE and KEYB commands add code page 850 (which should be the only Roman-based code page you need besides 437) and make CP850 your startup code page. In the KEYB command, you may substitute a different country code for US. Also add a line like this to your `CONFIG.SYS` file:

   ```
   DEVICE=C:\DISPLAY.SYS CON:=(EGA,437,(4,2))
   ```

3. Reboot your PC: Press the Ctrl, Alt, and Del keys all at the same time.

4. Type the DOS KEYB and CHCP commands to check that the new configuration has taken effect:

   ```
   C>chcp
   Active code page:  850
   C>keyb
   Current keyboard code: US Code page: 850
   Current CON code page: 850
   ```

5. Run MS-DOS Kermit:

   ```
   C>kermit
   ```

6. Issue a SET TERMINAL CHARACTER-SET command to tell Kermit which character set your host is using, and then issue the CONNECT command:

   ```
   MS-Kermit>set parity none
   MS-Kermit>set term bytesize 8
   MS-Kermit>set term char latin1
   MS-Kermit>connect
   ```

7. Enter special characters using Alt-key combinations (Alt-digit-digit-digit, see Table I-7), or specially labeled keys on a national keyboard (keys are labeled properly), or by loading a soft keyboard with the DOS KEYB command and referring to the appropriate keyboard template at the back of the DOS manual.

8. Using a text editor on the host, create a file with a few of these characters and then save it. Next, display the host file (for example, using the host TYPE command) and check that the special characters are displayed correctly.

9. If desired, customize your display and keys using Kermit's SET TRANSLATION INPUT and SET KEY commands.

File Transfer

MS-DOS Kermit can adapt itself to different host character sets during file transfer too, but only if it knows exactly what character sets are involved. MS-DOS does not record which file was created with which code page. This is something you have to remember for yourself so you can tell Kermit how to translate the file. If you use more than one code page, you can adopt some convention for naming your files, for example:

```
437JOHN.TXT
850VACE.TXT
```

or you can store them in separate directories.

During file transfer, Kermit translates character sets only when its FILE TYPE is TEXT. If you give the command SET FILE TYPE BINARY, or if the other Kermit sends a file to MS-DOS Kermit and includes the "binary" attribute, no translation is done. Kermit uses SET FILE TYPE TEXT by default.

International character set translation during file transfer is a recent addition to the Kermit protocol. You can use this feature only when the other Kermit also supports it. As of this writing, the following Kermit programs do (or will when they are released): C-Kermit 5A (for UNIX, VAX/VMS, and others); Macintosh Kermit 1.0 or later; and IBM 370 Mainframe Kermit 4.2 or later (for MVS/TSO, VM/CMS, and MUSIC).

Figure 13-1 File Transfer Character Set Translation

Basic Principles of International Text File Transfer

1. The Kermit program that is sending the file must be told the character set that the file is written in. The command is SET FILE CHARACTER-SET. It is required because, in general, the sending computer has no way of finding this out for itself. How many computers have you seen where the DIRECTORY command lists the file's character set? An IBM PC text file cannot contain a mixture of code pages; each file uses exactly one code page.[16]

2. The Kermit program that is receiving the file must be told what character set to use when writing the new file to disk. The command here is also SET FILE CHARACTER-SET. This command is necessary when there is a choice of character sets on the receiving computer.

3. *Both* Kermit programs must be told what character set to use for transferring the file. The command is SET TRANSFER CHARACTER-SET. Since there are hundreds of computer manufacturers in the world, and no one to control what codes they use to represent characters, it is impractical to require that Kermit have knowledge of *every* computer's codes. That's why a standard, intermediate transfer character set is used.

---

[16]Multilanguage documents are composed on the PC using proprietary word processing programs with private encodings that are, in general, not compatible with any other DOS application, let alone other kinds of computers. For purposes of file transfer, you should treat such files as *binary* files or ask the application to export them into some kind of standard or interchange format.

Once the three character sets have been specified, the sending Kermit reads characters from its local disk, translates them into the transfer character set, puts them in Kermit packets, and sends them to the receiving Kermit, which in turn translates these characters into its own local file character set and writes them to a disk file, as shown in Figure 13-1.

PC file character set ↔ Transfer character set ↔ Host file character set

```
MS-Kermit>set file char cp850   (PC Kermit...)
MS-Kermit>set transf char latin1
MS-Kermit>connect
$
$ kermit                        (Host Kermit...)
C-Kermit>set file char dec-mcs
C-Kermit>set transf char latin1
```

International Text File Transfer Commands

The only two commands you need for international text file transfer with MS-DOS Kermit are SET FILE CHARACTER-SET and SET TRANSFER CHARACTER-SET.

SET FILE CHARACTER-SET *name*

Use this command to tell MS-DOS Kermit which code page to use when sending a PC file, or which code page to translate an arriving file into. You may select any of the following code pages, even if you don't have them on your PC. The first five are supplied with most IBM PCs and PS/2s that have DOS 3.30 or later, and their codes are listed in Table I-7. The codes for Cyrillic code page 866 are listed in Table I-8.

CP437 The original IBM PC code page

CP850 The multilingual code page

CP860 Code page for Portugal

CP863 Code page for French Canada

CP865 Code page for Norway

CP866 Microsoft code page 866, the character set commonly used on PCs in the Soviet Union, similar to "Alternative Cyrillic." If you choose CP866, the transfer character set (see below) is automatically forced to be CYRILLIC.

The default file character set is your current code page, as shown by the DOS command CHCP. Setting MS-DOS Kermit's file character set does not change your PC code page.

SET TRANSFER CHARACTER-SET *name*

Give this command to tell MS-DOS Kermit which character set to use when communicating with the other Kermit program. When sending a file, MS-DOS Kermit announces this character set to the other Kermit, and it translates from the current FILE CHARACTER-SET (or the current code page if no file character set has been specified) into the transfer character set. When receiving a file, MS-DOS Kermit translates from this character set into the current file character set or code page. However, the transfer character set, if any, announced by the other Kermit takes precedence. Kermit's transfer character sets are:

LATIN1

ISO Standard 8859 Latin Alphabet Number 1 (see Table I-7). This character set is capable of representing any alphabetic character with any diacritical mark in any of the IBM code pages listed under the SET FILE CHARACTER-SET command, except the dotless i and the Greek and Cyrillic letters.

CYRILLIC

The ISO 8859-5 Latin/Cyrillic alphabet, capable of representing Bulgarian, Bielorussian, English, Macedonian, Russian, Serbian, and Ukrainian. If you choose CYRILLIC, the file character set is automatically forced to CP866.

TRANSPARENT

Do not translate characters at all. Use this option only when you want to keep the actual IBM PC codes intact when sending files to a different kind of computer and still retain Kermit's text file record format conversion. Similarly, do not translate characters when receiving files. For compatibility with previous MS-DOS Kermit releases, TRANSPARENT is the default transfer character set.

Examples:

```
MS-Kermit>set file character-set cp850       (Roman text)
MS-Kermit>set transfer character-set latin1

MS-Kermit>set transf cha cyrillic            (Cyrillic text)

MS-Kermit>set transf ch transp               (No translation)
```

The secret of successful international file transfer is knowing what file character set to use on the remote host. You should use a character set supported by the host version of Kermit and the applications on the host that will be using the file. Some hosts support 8-bit character sets like ISO Latin Alphabet 1, others support their own proprietary international character sets (like IBM mainframe Country Extended Code Pages), and others support only 7-bit codes like ASCII and the NRC sets. To find out what character sets your host Kermit program supports, read the documentation or use its built-in question-mark menus or help commands.

Examples of International Text File Transfer

The following examples show only the actual file transfer. By now, you should have grasped the concept that in order to transfer files, you have to set the appropriate parameters, connect to the other computer, and then log in; when you're finished using the other computer, you should connect back if necessary and then log out.

PC to VAX/VMS

PC code page 437 ↔ Latin-1 ↔ Latin-1

Here we send a document written in French on the PC using code page 437 to a VAX/VMS computer. The translation is CP437 to LATIN1. The transfer character set, LATIN1, is one that VAX/VMS computers know and support, so it can also be the file character set on the VAX.

```
MS-Kermit>set file character-set cp437
MS-Kermit>set transfer character-set latin1
MS-Kermit>connect               (Connect to the VAX computer)
$ kermit                        (Start Kermit on VAX computer)

C-Kermit>set file character-set latin1
C-Kermit>set transfer character-set latin1
C-Kermit>receive
Alt-X                           (Escape back to the PC)
MS-Kermit>send french.txt       (Send French text)

  (The file is transferred...)

MS-Kermit>set terminal vt320    (Must use VT320)
MS-Kermit>set term byte 8       (To view 8-bit characters)
MS-Kermit>set parity none       (Ditto)
MS-Kermit>set term char lat     (Note abbreviation)
MS-Kermit>connect               (Connect to the host)
C-Kermit>exit                   (Exit from host computer's Kermit)
$ type french.txt               (Look at the result)

Un serveur Kermit n'est qu'un programme Kermit qui fonctionne de
manière spéciale.  Son comportement est très similaire à celui
du Kermit ordinaire lorsque vous lui adressez la command RECEIVE.
Il attend la réception d'un message de la part de l'autre Kermit,
mais dans le cas du Kermit serveur, le message est une commande
disant ce qu'il faut faire, normalement envoyer ou recevoir un
fichier ou un groupe de fichiers.

$
```

To work with the file on VAX/VMS, your terminal type must be VT220 or VT320, and the terminal character set must be Latin-1.

PC to IBM Mainframe

PC code page 850 ↔ Latin-1 ↔ CECP37

Here we transfer a document in French, German, Italian, Norwegian, Icelandic, etc., or any combination of these, written on the PC using code page 850, to an IBM mainframe that uses Country Extended Code Page (CECP) 37, which contains the same special characters as CP850 but in different positions.

```
MS-Kermit>set file character-set cp850
Kermit-TSO>set transfer character-set latin1
MS-Kermit>connect                (Connect to the mainframe)
. kermit                         (Start Kermit on mainframe)
Kermit-TSO>set file character-set cp037
Kermit-TSO>set transfer character-set latin1
Kermit-TSO>receive
Alt-X                            (Escape back to the PC)
MS-Kermit>send saga.txt          (Send an Icelandic saga)
```

Although Kermit lets us transfer international text files between IBM mainframes and PCs, the mechanism used to view the file once it is on the mainframe is highly dependent on the host hardware and in some cases might not be possible at all. Check your local IBM mainframe documentation.

PC to UNIX with Only ASCII

PC code page 850 ↔ Latin-1 ↔ 7-bit ASCII

Here we have a file written in German on the PC, using either code page 437 or 850. For German, it doesn't matter which one is used since both of them include all the characters needed for German, in the same positions.

```
Um interaktiv mit dem Programm zu arbeiten, ruft man es von der
DOS-Kommando-Ebene auf, indem man den Programmnamen eingibt,
normalerweise "kermit".  Wenn der Eingabe-Prompt "MS-Kermit>"
erscheint, können nacheinander beliebig viele Kermit-Befehle
eingegeben werden, bis man fertig ist und das Programm verlassen
möchte.  Die Befehle EXIT oder QUIT bringen einen zurück zu DOS.
```

Now we want to transfer this file to a UNIX system that can't use 8 bits, but still not lose any information. Luckily, there is a rule for removing umlauts (diaereses) from German: Each vowel that has an umlaut can be replaced by the same vowel followed by the letter e, for example ä becomes ae. The other German special character, ß, can be replaced by ss. C-Kermit performs these translations if you tell it the language is German:

```
MS-Kermit>set file character-set cp850
MS-Kermit>set transfer character-set latin1
MS-Kermit>connect                (Connect to UNIX)
% kermit                         (Start Kermit there)
C-Kermit>set language german     (Specify the language)
C-Kermit>set file character-set ascii
C-Kermit>set transfer character-set latin1
C-Kermit>receive
Alt-X                            (Escape back to PC)
MS-Kermit>send gisbert.txt       (Send the German file)
MS-Kermit>connect                (Look at the results)
C-Kermit>exit
% cat gisbert.txt

Um interaktiv mit dem Programm zu arbeiten, ruft man es von der
DOS-Kommando-Ebene auf, indem man den Programmnamen eingibt,
normalerweise "kermit".  Wenn der Eingabe-Prompt "MS-Kermit>"
erscheint, koennen nacheinander beliebig viele Kermit-Befehle
eingegeben werden, bis man fertig ist und das Programm verlassen
moechte. Die Befehle EXIT oder QUIT bringen einen zurueck zu DOS.
```

Notice the change in the words können, möchte, and zurück.

Kermit may also do the same kind of conversion for other languages that have similar rules. For example, in certain Scandinavian languages, å can be written as aa. For languages that do not have these rules, Kermit simply strips the diacritical marks and leaves the letters bare. For example, French pâté becomes simply “pate.”

PC to PC

There are two methods of PC-to-PC file transfer, depending on whether the two PCs are using the same code page.

PC code page 850 ↔ PC code page 850

If both PCs are using the same code page (a common case), all translations can be skipped. This brings an added benefit: Any mixture of text and binary files can be transferred together in a single operation:

```
MS-Kermit>set file type binary
MS-Kermit>cd \dan
MS-Kermit>send *.*
```

If you are running MS-DOS Kermit 3.0 or later on the receiving PC, you don’t have to give a SET FILE TYPE BINARY command to it; the sending Kermit tells the receiving Kermit automatically.

PC code page 865 ↔ Latin-1 ↔ PC code page 860

Suppose you are a Norwegian in Oslo who uses a PC with IBM's Norwegian keyboard and code page 865, and your company has a branch office in Lisbon, Portugal that uses PCs with Portuguese keyboards and CP860, and you want to send a report you have written in Norwegian to your friend who is stationed in Lisbon. The two IBM PC code pages do not have the same characters in them, as you can see from Table I-7: The Norwegian characters are missing from the Portuguese code page and vice versa.

Without Kermit's translations, your Norwegian Ø would become Ù, and your other special characters would change in more subtle ways. When translating between two Roman-based character sets that don't have equivalent characters, Kermit makes a best attempt at a sensible result, in this case removing the accents from Norwegian letters that don't appear with the same accents in Portuguese, e.g. Ø becomes simply O.

PC to Macintosh

The Macintosh character set supports most of the same international characters as a PC with Code Page 850, but (of course) uses completely different code values. You can transfer PC files in any code page with the Macintosh, but CP50 most closely matches the Macintosh character set. If you have a file in CP437 containing Greek or box-drawing characters, these will be lost—or, more precisely, they will turn into question marks after transfer.

PC code page 850 ↔ Latin-1 ↔ Apple Macintosh

Let's say you have dialed up a remote Macintosh which has been set up to run Mac Kermit 1.0 or later in server mode, and has been told (by clicking on the appropriate options in the File menu) to use Latin-1 as its transfer character set. Your PC uses code page 850. Just tell Kermit on your PC to SET TRANSFER CHARACTER-SET LATIN1 and you can SEND and GET as many files as you like.

Shortcuts

Readers who live in countries where English is not the primary language might feel that all this is a lot of effort to transfer files in their own language. The effort can be reduced or entirely eliminated by using Kermit's defaults and other shortcuts.

- Type the MS-DOS Kermit command SHOW FILE to see if your current file character set is the one you intend. If so, no SET FILE CHARACTER-SET command is necessary.
- When MS-DOS Kermit sends a file to another Kermit program, it notifies the other Kermit of the transfer character set *if* the other computer supports a feature called

attribute packets, and it also supports character-set translation. If this is so, it is unnecessary to give the SET TRANSFER CHARACTER-SET command to the receiving Kermit program. So far, all the Kermit programs that can do character set translation during file transfer also have the attribute feature.

- Similarly, when receiving a file from another computer, MS-DOS Kermit automatically recognizes the transfer character set *if* the other computer precedes the file with an attribute packet that contains this information.

- See Chapter 14 for additional shortcuts that apply to Kermit commands in general. For example, if you always use the same character sets, put the appropriate SET commands in your Kermit initialization file. If you switch MS-DOS Kermit among different hosts, define macros for each host that include the appropriate character set definitions.

The moral is that you can leave out many of the character-set related commands with a good chance that everything will still work correctly. Here, for example, is all you need to send an Italian-language text file from MS-DOS Kermit to IBM mainframe Kermit 4.2 or later:

```
Kermit-CMS>set file char cp500        (On the mainframe...)
Kermit-CMS>receive
Alt-X
MS-Kermit>set transf char latin1      (On the PC...)
MS-Kermit>send verona.txt
```

The PC knows what its current code page is, translates from it into LATIN1, and tells CMS Kermit to expect LATIN1. CMS Kermit translates from LATIN1 to code page 500. If you put the following command in your PC's `MSKERMIT.INI` file:

```
set transfer character-set latin1
```

and put these commands in your CMS Kermit initialization file on the mainframe:

```
set file character-set cp500
set transfer character-set latin1
```

then you can send and receive files written in most Roman-alphabet-based languages with no more effort than it takes to transfer plain ASCII text:

```
Kermit-CMS>receive
Alt-X
MS-Kermit>send verona.txt
```

Try it yourself!

Chapter 14

Macros, Command Files, and Scripts

In this chapter, you learn about the customizations that can perfectly suit MS-DOS Kermit to your communications environment, no matter how complex, and about the shortcuts you can take to invoke any collection of settings or actions automatically when Kermit starts up, by typing a single command or even by pressing a single key. All the settings and options described in the preceding chapters can be collected and condensed into files, commands, or keys that are so concise and easy to use that, once you have set them up, you can happily forget nearly all that you have learned so far.

MS-DOS Kermit is a powerful and flexible communication program. Because it gives you control over virtually every aspect of its operation, it can be adapted to nearly any situation. That's why there are so many commands! But its wealth of commands can make Kermit intimidating to the casual user and cumbersome even for the most experienced. Continue reading and you will learn how to make Kermit much easier to use, both for yourself and others. You will even be able to set up complicated "canned" procedures that the computerphobes in your office won't be afraid to use.

Command Macros

In Chapter 7, we looked at communication parameters like speed, parity, duplex, flow control, and handshake. If you must switch your PC frequently between two computers that are very different from each other—say, an IBM mainframe and a VAX/VMS system—would you want to type five or ten long SET commands each time you switched from one to the other? Of course you wouldn't.

Suppose you've read all the previous chapters and learned how to adapt your connection to the different types of host computers. Each time you used the IBM mainframe (in linemode), you would issue all these SET commands:

```
MS-Kermit>set flow none          (Communication settings)
MS-Kermit>set parity mark
MS-Kermit>set handsh xon
MS-Kermit>set duplex half
MS-Kermit>set speed 4800
MS-Kermit>set block 3            (Protocol settings)
MS-Kermit>set window 1
MS-Kermit>set rec pack 1000
MS-Kermit>set key \270 \8        (Backspace key setting)
```

And to switch to the VMS system, you would issue these SET commands:

```
MS-Kermit>set parity none        (Communication settings)
MS-Kermit>set handshake none
MS-Kermit>set flow xon/xoff
MS-Kermit>set duplex full
MS-Kermit>set speed 9600
MS-Kermit>set rec pack 200       (Protocol settings)
MS-Kermit>set window 8
MS-Kermit>set block 1
MS-Kermit>set key \270 \127      (Backspace key setting)
```

To switch back to the IBM mainframe, you'd have to enter the first group again, and to switch back to the VAX, you'd have to enter the second group again. And so on. If you need further motivation to continue reading, do this ten times.

Defining Macros

You can make up short slang words to refer to collections of settings and other commands. These words and the Kermit commands you assign to them are called *macros*. Once defined, a macro is used just like any other Kermit command. For example, let's create one macro for our IBM mainframe settings and a second one for VAX/VMS:

```
define ibm set parity mark, set handshake xon, set flow none,-
 set duplex half, set speed 4800, set receive packet-length 1000,-
 set window 1, set block 3, set key \270 \8
define vms set parity none, set handshake none,set flow xon/xoff,-
 set duplex full, set speed 9600, set receive packet-length 250,-
 set window 8, set block 1, set key \270 \127
```

The command for creating macros is DEFINE. The first word after DEFINE is the macro name. You can pick any name you like. After the macro name comes its definition, which is a list of Kermit commands separated by commas. The definition can be up to 1000 characters long. You can use any valid abbreviations for the Kermit commands you include in your definition, and this is a useful (but cryptic) space-saving technique:

```
def vms set pa n, set h n, set fl x,-
 set dup f, set sp 9, set rec pack 250,-
 set wi 8, set bl 1, set k \270 \127
```

This definition has 99 characters, versus 174 in the original unabridged version. Both do exactly the same thing. Dash characters can be used as shown to continue the long macro definition (or any other long command) onto multiple lines. The dash must be the last character on each continued line.

You can see the definition of a macro by typing the SHOW MACRO command at the `MS-Kermit>` prompt:

```
MS-Kermit>show macro vms
 VMS = set pa n<cr>
  set h n<cr>
  set fl x<cr>
  set dup f<cr>
  set sp 9<cr>
  set rec pack 250<cr>
  set wi 8<cr>
  set bl 1<cr>
  set k \270 \127<cr>
```

`<cr>` means carriage return (the character produced when you press the Enter key). This is what Kermit translates the comma between the commands into. All commas in macro definitions get this treatment, so if you want to include an actual comma in the definition, use `\44` (the ASCII value of comma). If you type SHOW MACRO alone, Kermit displays all your macros. If you type SHOW MACRO C, Kermit shows all the macros that begin with the letter C.

When you invoke a macro by typing its name at the `MS-Kermit>` prompt, all the Kermit commands in the definition are executed in order:

```
MS-Kermit>ibm
```

You can also give the command DO before the actual macro name if this verb helps you to remember that an action is taking place:

```
MS-Kermit>do ibm
```

In either case, you might not believe that these parameters were set because there is no message verifying this. When in doubt, you can use the SHOW COMMUNICATIONS command to check the settings:

```
MS-Kermit>ibm
MS-Kermit>show communications
  Duplex: half          Parity: mark
  Handshake used: ^Q    No flow control used
```

(^Q means Ctrl-Q, which is the character associated with Xon.) Now switch to the VAX/VMS settings and verify them:

```
MS-Kermit>vms
MS-Kermit>show communications
  Duplex: full          Parity: none
  Handshake used: none  Flow control: Xon/Xoff
```

Noisy Macro Example

Just as you can set up macros to switch between the different types of computers you communicate with, you can also create macros for different quality connections. Suppose you make a lot of modem calls for transferring files. Sometimes you get a clean, noise-free connection. At other times, you get a very noisy connection, resulting in garbage characters on your screen:

```
Wel}}}} to VAp+=MS              (Welcome to VAX/VMS)
Us||na^e:                       (Username:)
```

Recall the scenario from Chapter 9 where you had a clean communications line at work and a noisy one at home. On a normal connection, you could keep the packet length at 94, give up after 4 retries, and use the least amount of error checking. On a noisy line, you could reduce the packet size to 40 characters instead of 94, allow 20 retries instead of only 4, apply Kermit's strongest error check on the packets, and, to compensate for the reduction in efficiency, activate the sliding windows protocol with a fairly large window size. And, on a very clean connection, you could try using long packets, a medium-strength error checking technique, and a small sliding window size to achieve maximum efficiency. Here are three macros you can use:

```
def normal set rec pac   94, set ret  5, set block 1, wet window 1
def noisy  set rec pac   40, set ret 20, set block 3, set window 8
def clean  set rec pac 1000, set ret  4, set block 2, set window 2
```

Now when you make your connection, you can type the commands NORMAL, CLEAN, or NOISY depending on what your screen looks like. Use the SHOW PROTOCOL command to check your settings:

```
MS-Kermit>clean
MS-Kermit>show protocol
  Receive packet size: 1000   Packet Retry limit: 4
  Sliding window slots:   2   Block-check:        2
```

Macros with Arguments

So far we've just used macros to combine a list of Kermit SET commands. Let's see what else macros can do.

Just like real Kermit commands, macros can have arguments. Can not! Can so! Can not! Can so! No, not that kind of argument. In computer jargon, an *argument* is like the object of a verb. The verb tells what is being done, and the object tells who or what it is being done *to*. In a macro, the macro name tells Kermit what to do, and the arguments tell Kermit how to do it, who to do it to, when to do it, how many times to do it, and so on.

For example, even though Kermit already has several DOS functions built into it (TYPE, DELETE, DIRECTORY), you can define macros that use Kermit's RUN command to include additional DOS commands so you can stay in Kermit all day:

```
def more run more < \%1          (DOS MORE command)
def rename run ren \%1 \%2       (DOS REN command)
def copy run copy \%1 \%2        (DOS COPY command)
def edit run edlin \%1           (For editing files)
```

With these macros, you can issue DOS-like commands directly from the `MS-Kermit>` prompt:

```
MS-Kermit>more juri.txt
MS-Kermit>rename andrei.txt misha.txt
MS-Kermit>copy marina.c kostya.c
MS-Kermit>edit shamil.txt
```

The DOS MORE command is like TYPE, but it pauses after each screenful. It does not accept a filename directly, as TYPE does. The filename must be given using the DOS input redirection symbol, `<`. Kermit's MORE macro, defined above, supplies this symbol itself. These macros illustrate the use of Kermit's positional arguments. In the RENAME macro:

```
RENAME                           (is the macro name)
RUN REN \%1 \%2                  (is the macro definition)
\%1 \%2                          (are the arguments)
```

`\%1` is the first argument, `\%2` is the second, and so on, up to `\%9`. These arguments change into whatever words (in this case, filenames) you type after the macro name in a *macro invocation*, such as this one for the RENAME macro:

```
MS-Kermit>rename boring.txt oofa.txt
```

`\%1` is replaced by `boring.txt`, and `\%2` is replaced by `oofa.txt`. The arguments `\%3` through `\%9` contain nothing, because the invocation included only two arguments. The argument `\%0` contains the name of the macro, in this case RENAME. To illustrate:

```
MS-Kermit>define showargs echo "\%0"[\%1][\%2][\%3][\%4]
MS-Kermit>showargs hi there
"SHOWARGS"[hi][there][][]
```

In this example, `\%0` is the macro name SHOWARGS, `\%1` is "hi", `\%2` is "there", and the remaining arguments are empty.

If you call a macro from within another macro, a new set of arguments is created, and the previous set is preserved. *This is changed from MS-DOS Kermit version 3.0, in which there was only one set of* \%1–\%9 *arguments, shared by all macros.* To illustrate:

```
MS-Kermit>define x1 echo [\%1][\%2], x2 sleep, echo [\%1][\%2]
MS-Kermit>define x2 echo (\%1)(\%2), define \%2 tight
MS-Kermit>x1 good evening
[good][evening]
(sleep)()
[good][evening]
```

In MS-DOS Kermit 3.0, the X1 macro's was changed behind its back:

```
[good][evening]
(sleep)(evening)
[sleep][tight]
```

This change makes it impossible for macros to interfere with each others' arguments, which could produce unwanted effects:

```
MS-Kermit>define erase delete \%1
MS-Kermit>more resume.txt
MS-Kermit>erase
```

In earlier versions, the ERASE command would delete your résumé because the \%1 value from the MORE command was carried over into the ERASE command. In 3.10 and later, your résumé is safe.

Macros on Keys

Now that you know how to define a macro and invoke it by typing a single word (possibly followed by arguments), you will be glad to learn that there is also a way to invoke a macro in a single keystroke.

You can assign any macro, or combination of macros, to any key. Push that key during CONNECT mode, and Kermit executes the macro (remember, key translations are only active during CONNECT mode). If you want to remain in CONNECT mode after pressing the key, be sure to include CONNECT as the final command in the macro's definition.

Here is an example in which the PC's F1 (\315) and F2 (\316) keys are set up to change the screen writing direction. This is handy for entering Hebrew or Arabic words into an English or French document, or vice versa (assuming your host computer's word processor is set up for this capability and you have a Hebrew or Arabic code page loaded):

```
define left set terminal direction left, connect
set key \315 {\kleft}
define right set term dir right, connect
set key \316 {\kright}
```

The macro names are LEFT and RIGHT. When assigning macros to keys, you must precede the macro name by \k and surround the whole thing with braces.

Here is another example that lets you receive files from a remote Kermit program by just pressing the F2 key instead of escaping back, typing RECEIVE, and then typing CONNECT again when the transfer is done:

```
define rcvkey receive, connect
set key \316 {\Krcvkey}
```

The macro name is RCVKEY. When F2 is pressed during CONNECT mode, MS-DOS Kermit goes into file receiving mode, and when the transfer is complete, it connects back to the remote computer.

Command Files

We've seen how to save typing by defining keys and macros, but typing the definitions is a lot of work in itself, and once you exit from the MS-DOS Kermit program, your macro definitions are forgotten. So you need some way to save macro definitions for future use. For this reason (among others), Kermit is able to execute commands not only from the keyboard but also from disk files. If you give the command TAKE followed by a filename, Kermit executes the commands that are in the file:

```
MS-Kermit>take defs.tak
```

This file could include all of your macro and key definitions. If you do not include a disk or directory name in the TAKE file specification, Kermit looks in the current directory. If the file is not there, it looks in the directories listed in your DOS PATH (if you have one). Kermit command files must contain only lines of ordinary text—no special effects like underlining, boldface, or italics, and no pagination or other information that word processors might add. If you create a text file with a word processor, be sure to save it "text-only."

Here is a handy use for a command file. Suppose you want to send many files to another computer, but since they are scattered all over the place, there is no way you can specify them in a single SEND command. But you don't want to sit at your PC for hours typing separate SEND commands for each. If you put the receiving Kermit in server mode, you can collect your SEND commands into a command file called, say, LUNCH.TAK:

```
cd \amy
send lavender.*
send \peter\dragons.txt
cd \joann
send *.*
send a:\kermit\mskermit.ini
bye
```

Now use this command file to send all of your files while you eat lunch:

```
MS-Kermit>connect              (Connect to the host computer)
$ kermit                       (Run Kermit on the host)
C-Kermit>server                (Put it in server mode)
```

```
Alt-X                          (Escape back to the PC)
MS-Kermit>take lunch.tak
```

Let's create a command file using EDLIN, DOS's standard, no-frills line editor:

```
C>edlin defs.tak               (Create file DEFS.TAK)
*i                             (Insert text)
       *1:set key \315 I just typed F1.\13
       *2:set key \316 Did you type F2?\13
       *3:set key \317 F3 is my favorite.\13
       *4:Ctrl-C
*e                             (Save file; exit EDLIN)
C>
```

To execute the file, run the MS-DOS Kermit program and give the TAKE command:

```
C>kermit
MS-Kermit>take defs.tak
MS-Kermit>connect
```

Now you're connected to the host, and your SET KEY commands take effect:

- Type the F1 key.
  (You should see the sentence:) `I just typed F1.`
- Type the F2 key.
  (You should see the sentence:) `Did you type F2?`
- Type the F3 key.
  (You should see the sentence:) `F3 is my favorite.`

Of course, these sentences won't make any sense to the host computer, but I am sure you can think of ones that will.

Initialization Files

What if you forget to issue your TAKE command each time you run Kermit? If you want to have certain settings and definitions in effect all the time, you can collect them into an *initialization file* called `MSKERMIT.INI` so they will be executed automatically every time you run Kermit. On other kinds of computers, Kermit initialization files have other names, like `CKERMIT.INI` for VAX/VMS C-Kermit or `.kermrc` for UNIX Kermit.

The `MSKERMIT.INI` file must be in your current directory or in your DOS PATH, as described in Chapters 2 and 3, or else Kermit won't find it. To reassure yourself that Kermit has executed your initialization file, you can include an ECHO command in it, which displays text on your screen. For example, the line:

```
echo Happy Birthday!\13
```

gives you the message `Happy Birthday!` every time Kermit starts up.

Here's a sample MSKERMIT.INI file for an IBM PC that is connected to an IBM mainframe on port 1 and uses a modem on port 2 to dial up a VAX computer.

```
COMMENT - Set up modem parameters for dialing up VAX systems...
  set port 2                    ; Select port 2
  set speed 1200                ; Port 2 speed is 1200
  set parity none               ; Port 2 parity is none
  set duplex full               ; Port 2 duplex is full
  set flow xon/xoff             ; Port 2 flow control
  define modem set port 2, set key \270 \127

COMMENT - Set up direct line for IBM mainframe in linemode...
  set port 1                    ; Select port 1
  set speed 9600                ; Port 1 speed is 9600
  set parity mark               ; Port 1 parity is mark
  set duplex half               ; Port 1 duplex is half
  set flow none                 ; Port 1 no flow control
  define ibm set port 1, set key \270 \8
  do ibm

COMMENT - General settings...
  set file collision rename     ; Protect my files
  set terminal cursor block     ; Use a big cursor
  set terminal screen reverse   ; Use reverse video
  echo Smile!
```

MS-DOS Kermit remembers port-related settings for each port, so all you have to do to switch between them is SET PORT 1, SET PORT 2. But since the IBM and VAX systems use a different code for backspacing, and this is not a port-related parameter, macros are defined to SET PORT and SET KEY for each system.

Command files may include COMMENT lines for documentation, and commands in TAKE files can also have trailing comments that start with a semicolon (;), as shown. Trailing comments cannot be used with interactive commands or within macro definitions.

TAKE files may include TAKE commands. If you wish, you can have a TAKE file that TAKEs many other TAKE files or a TAKE file that TAKEs a TAKE file, which TAKEs another TAKE file, and so on to a depth of about 20. The commands that Kermit reads from the file are not normally displayed on your screen, but you can ask Kermit to show them to you by giving the command SET TAKE-ECHO ON.

Command files can be as short and simple or as long and complicated as you want to make them. If you have installed Kermit on your hard disk according to the instructions in Chapter 2, you can put command files in your \KERMIT directory (which is in your DOS *PATH*), so Kermit's TAKE command can find them no matter what your current disk or directory might be.

Script Programming

Gaining access to another computer can be a tedious and repetitive process: dialing your modem, waiting for the connection, typing your access codes, waiting for the system prompt—day after day the same thing. Aren't we supposed to be using computers to automate the tedious and repetitive processes for us?

Perhaps you're thinking that you could use Kermit's TRANSMIT command to send all the characters that you would normally type during this process out the serial port. Good idea, but it usually won't work. The reason it usually won't work is that you have to synchronize the characters you type with the prompts and responses of the computer or modem. Remember, things don't always go smoothly—sometimes you have to make decisions. For example, if your modem tells you `NO ANSWER` instead of `CONNECT 2400`, then you would (or should) *decide* not to type the login sequence.

If Kermit can trick the modems and computers into thinking that you are sitting there in person typing at them at each moment, then it is performing a kind of "human emulation." How can we coordinate Kermit's actions with the events that Kermit observes through the communication port?

As soon as we start talking about a computer making decisions, we're talking about *programming*. But don't let this word scare you if you're not a programmer. Kermit programs are called *scripts*. A script is just a list of Kermit commands, in either a command file or a macro or any combination of the two, that contains some special human emulation commands—mimicking what you would type, looking at the computer's response and deciding what to do, and trying things over again when they don't work. In other words, doing what you would do (but more patiently).

To automate a procedure, you must tell MS-DOS Kermit to issue all the commands that you would normally type when interacting with Kermit itself, the modem, and the remote computer. The one difference, which often leads to confusion, is that you must do all this *without* giving the CONNECT command. CONNECT reads text only from the keyboard, not from a command file or macro definition. A command file like:

```
set speed 9600
connect
atdt8765432
login peppi
```

would not work: The CONNECT command would be executed, and Kermit would wait for you to start typing. When you finally escaped back, Kermit would try (and fail) to execute the next two lines as Kermit commands. Only put a CONNECT command in a script when you really want to step in and take over the interaction yourself.

The INPUT and OUTPUT Commands

Two special commands, INPUT and OUTPUT, take the place of CONNECT for interacting with the modem or the remote computer:

INPUT

What you would expect to see on your screen. The INPUT command tells Kermit to wait a certain number of seconds for the specified sequence of characters to arrive at the communication port. The format of the command is:

INPUT *n characters*

where *n* is the number of seconds to wait, and *characters* are what Kermit is to look for during those seconds. For example:

```
input 10 Username:
```

This command tells Kermit to wait up to 10 seconds for the VAX/VMS `Username:` prompt to arrive. If you leave out the number, Kermit waits its default number of seconds (one, but you can change it):

```
input Username:
```

The INPUT command "succeeds" if the matching characters show up within the timeout interval, and it "fails" otherwise. If you become impatient during the timeout interval, you can force the INPUT command to fail immediately by pressing any key on the PC keyboard.

OUTPUT *characters*

What you would normally type. The OUTPUT command tells Kermit to send the given characters out the communication port to the modem or the remote computer:

```
output I am not really typing this...
```

INPUT and OUTPUT text can contain any sequence of 7-bit ASCII characters. Printable ASCII characters can be included literally, as shown above. Control characters can be inserted by giving their numeric ASCII code values, preceded by a backslash (see Tables 17-2 and I-5); for example:

```
output atdt7654321\13
```

sends a tone dialing command to a Hayes modem (use ATDP for pulse), followed by a carriage return (`\13`) (as if you pressed the Enter key). International characters are not translated; they must be represented by backslash codes denoting their code values in the remote computer's character set. For example:

```
output Gr\252\223e
```

sends the German word "Grüße" to a computer that uses the Latin-1 alphabet (listed in Table I-7). The OUTPUT command only sends the characters you specify, so be sure to include `\13` wherever you would press the Enter key. The OUTPUT string can also contain a special symbol, `\b`, to send a BREAK signal.

The behavior of the INPUT command is controlled by the SET INPUT command:

SET INPUT CASE IGNORE *or* **OBSERVE**

Tells whether to pay attention to alphabetic case during INPUT. Case is IGNOREd by default. This means that the command `INPUT 5 LOGIN` would succeed if `LOGIN` or `Login` or `lOgIn` appears. To require an exact case match, use:

```
set input case observe
```

SET INPUT DEFAULT-TIMEOUT *seconds*

Tells how long to wait for the INPUT string if the waiting time is left out of the INPUT command. For example:

```
set input default-timeout 10
input Username:
```

will cause the INPUT command to wait 10 seconds for the `Username:` prompt. The default timeout is one second if you don't change it with this command.

SET INPUT ECHO ON *or* **OFF**

Tells whether characters read by the INPUT command should echo on your screen. INPUT ECHO is ON by default, so you can watch. Turn it off for "silent running:"

```
set inp echo off
```

Characters that are INPUT and then displayed on the screen are *not* passed through Kermit's terminal emulator, so screen-oriented host output might look garbled. This problem can be somewhat alleviated by using IBM's `ANSI.SYS` console driver.

SET INPUT TIMEOUT-ACTION PROCEED *or* **QUIT**

Tells what Kermit should do if an INPUT command fails to read the specified string within the given amount of time. PROCEED means just go on to the next command, and QUIT means exit from the current macro or command file immediately, without executing the remaining commands. The default setting is PROCEED.

In simple cases, INPUT and OUTPUT are the only special commands necessary for conducting an automated dialog. For example, if the following sequence was stored in a command file called `VAX.SCR`:

```
set speed 2400                  ; Communication settings
set input timeout quit          ; Give up if INPUTs fail
output ATD5554321\13            ; Dial the Hayes modem
input 20 CONNECT                ; Wait 20 secs for CONNECT message
output \13                      ; Send a carriage return
input 10 Username:              ; Wait 10 secs for host prompt
output cmg\13                   ; Send my username
connect                         ; I'll do the rest myself
```

you could dial up and log in to your VAX simply by giving the command:

```
MS-Kermit>take vax.scr
```

And often it will actually work. When it doesn't, one of the INPUT commands will fail, and the `MS-Kermit>` prompt will return. *HINT:* The Hayes modem dialing command used in this example is simply ATD. This tells the modem to dial the number using its default dialing method (tone or pulse). If this doesn't work for you, use ATDT (Tone) or ATDP (Pulse) explicitly.

The PAUSE and CLEAR Commands

Why should our simple script fail? Perhaps a certain amount of time is needed after the modem's `CONNECT` message for the communication channel to become completely clear. If the carriage return (`\13`) that is supposed to wake up the VAX is sent during this time interval, it could be lost. Maybe the dialing command that was sent to the modem echoed back and confused matters. The PAUSE and CLEAR commands can help:

PAUSE *seconds*

Does nothing for the specified number of seconds, for example:

```
pause 10
echo Hello, it's ten seconds later!
```

If the seconds are omitted, Kermit pauses for one second. The pause is interrupted if you type anything on the keyboard. PAUSE "succeeds" if the time interval expires, and "fails" if it is interrupted. If characters arrive at the communication port during the PAUSE interval, they are stored in the INPUT buffer and, if SET INPUT ECHO ON is in effect, they are also displayed on the screen.

CLEAR

Clears the communication port input buffer. If you know that characters have arrived from the host that are not important to the script, or that could confuse it, use this command to discard them.

Here is the script with the PAUSE and CLEAR commands added:

```
set speed 2400                  ; Communication settings
set input timeout quit          ; Quit if any INPUT fails
output ATD5554321\13            ; Dial the Hayes modem
clear                           ; Clear away ATD echo
input 20 CONNECT                ; Wait 20 secs for CONNECT message
pause 2                         ; Give 2 secs for settling
output \13                      ; Send a carriage return
input 10 Username:              ; Wait 10 secs for host prompt
output cmg\13                   ; Send my username
connect                         ; I'll do the rest myself
```

The STOP, IF, and GOTO Commands

We have reduced the chances of failure but not eliminated them. Pitfalls still remain:

- The modem is not turned on.
- The modem is not connected to the telephone, or the telephone is broken.
- The modem call is not answered.
- The VAX is down.

No matter how powerful Kermit is, it can't plug in your modem, fix your phone, or revive your VAX. But it can give you informative messages so you can correct the problems yourself. For this, you'll need some more new commands:

GOTO *label*

Instead of executing the next command, Kermit goes to the specified script label and begins executing commands there. A label is on a line by itself, flush against the left margin, and begins with a colon (`:`). Here is an example:

```
echo Skipping the next message
goto last
echo You should not see this!
:last
echo This is the last message
```

The label can be either before or after the GOTO statement. If it is before, you can create an infinite loop (don't try this at home folks!):

```
:loop
echo again and
goto loop
```

This prints "again and again and again and" forever. (You can interrupt this or any script program by typing *`Ctrl-C`*. You might need to do this several times.)

STOP

Exits from the script command file (or macro) immediately without executing any more commands and returns to the `MS-Kermit>` prompt.

IF SUCCESS *command*

Executes the *command* only if the preceding command succeeded, for example:

```
input 10 login:
if success goto username
```

IF FAILURE *command*

Executes the *command* if the preceding command failed, for example:

```
input 10 login:
if failure stop
```

Let's apply these new commands to our example:

```
set speed 2400            ; Communication settings
set input timeout proceed ; Check failures with IF
output AT\13              ; See if modem is there
input 2 OK                ; Should say "OK"
if success goto dial      ; If so, go and dial
echo Please connect or turn on your modem!\13
stop

:dial
output ATD5554321\13      ; Dial the Hayes modem
input 20 CONNECT          ; Wait 20 seconds for CONNECT message
if success goto connected ; So far so good
echo Dialing failed - try again later.
stop

:connected
pause 2                   ; Settling time
output \13                ; Send a carriage return
input 10 Username:        ; Wait 10 seconds for Username:
if success goto login
echo No VAX login prompt - try again later.
hangup
stop

:login                    ; Everything worked
output cmg\13             ; Send my username
connect                   ; I'll type the password myself!
```

Now we're getting a little more "user-friendly." This script should always succeed or else tell you why it didn't.

The REINPUT, SET ALARM, SET COUNT, and IF COUNT Commands

How much more intelligence can we add to our little dialing script? A lot! Let's consider two more cases that you would handle routinely if you were making the connection manually:

- The modem says `BUSY` rather than `CONNECT`. Our script gives up right away, without a fight. But most people would wait a while and then dial again. And most people would also lose patience after five or ten tries, and then they would give up and find something else to do.
- Dialing is at 2400 bits per second (bps), but the other modem answers at 1200 bps, so the local modem drops down to 1200 bps also and gives the message `CONNECT 1200`. But Kermit's speed is still set to 2400. You or I would notice this and adjust Kermit's speed, but our script program is blissfully unaware that this has happened and will try to log in to the VAX at the wrong speed.

To let Kermit imitate our reasonable behavior, we need new commands for repeating groups of script commands a certain number of times, for checking on elapsed time, and for reexamining the INPUT buffer. Here are the commands we will need:

REINPUT *seconds characters*

Just like INPUT, but instead of reading characters as they arrive at the serial port, the REINPUT command rereads the ones that previously arrived. This way, messages from the modem or the host can be scanned several times for different words or phrases, like `CONNECT 2400` or `CONNECT 1200`.

SET ALARM *seconds*

Start a timer for the specified number of seconds.

IF ALARM *command*

Execute the *command* if the number of seconds given in the most recent SET ALARM command has passed. Example:

```
set alarm 60
pause 60
if alarm echo You're so patient.
if not alarm echo You typed something!
```

SET ALARM is especially useful for testing whether an INPUT command failed because its timer expired or because the user interrupted it from the keyboard.

```
set alarm 60
input 60 Login:
if success go gotlogin
if not alarm echo Please don't interrupt!
```

SET COUNT *number*

Put a limit on the number of times a statement or group of statements will be executed, for example:

```
set count 5
```

IF COUNT *command*

Subtract one from COUNT. If the result is greater than zero, Kermit executes the *command*, for example:

```
if count goto loop
```

Here is an example of a counted (finite) loop:

```
set count 3
:again
echo Hello
if count goto again
echo Goodbye
```

This script says "Hello" three times and then says "Goodbye."

Overuse of IFs, GOTOs, and labels can make a program unclear, hard to read, and therefore prone to programming errors as you modify it. Since our major use of these features so far is to issue a message and stop if there is an error, we can define a macro to do this and avoid labels and GOTOs in these cases:

```
define errstp echo Error: \%1, hangup, stop
```

This macro, named ERRSTP (short for ERROR STOP), echoes its argument, `\%1` (the first positional argument, remember?). The HANGUP command closes the telephone connection, and then the STOP command terminates the script and returns immediately to the `MS-Kermit>` prompt.

Positional arguments are assigned one word at a time. A statement like:

```
errstp Please turn on your modem.
```

would assign only the word "Please" to the argument `\%1`, and so the error message would just be `Error: Please`. To allow more than one word to be assigned to a single macro argument, Kermit lets you group words within curly braces when using them after a macro name, for example:

```
errstp {Please turn on your modem.}
```

resulting in `Error: Please turn on your modem`.

By adding loops to our script, we can make it keep dialing until the other computer answers, up to whatever limit we give in the SET COUNT command. And by adding the REINPUT command, we can process the modem's messages more intelligently. While we're at it, we should remove the part about logging in to the VAX and connecting. This way the script will be an all-purpose Hayes modem dialer, which you can invoke from within any command file, script, or macro, or from the MS-DOS Kermit command prompt. To be truly general, however, the dialing script needs one more modification: It must be able to call *any* telephone number, not just 5554321. So in the Hayes ATD dialing command, we will use the `\%1` argument to stand for the phone number.

Let's call our new script file `HAYES.SCR`. It can be executed by typing `take hayes.scr` at the `MS-Kermit>` prompt. Now how can we tell the file what telephone number to dial? Easy—define a DIAL macro that TAKEs the command file:

```
define dial take hayes.scr
```

When you give a command like "dial 555-1212", the macro automatically assigns its first argument, the phone number, to the first positional argument, `\%1`.

```
MS-Kermit>dial 555-1212
```

Now, when the macro TAKEs the `HAYES.SCR` command file, `\%1` contains the phone number so the statement `output ATD\%1\13` will call the given number.

```
def errstp echo Error: \%1,hang,stop
set speed 2400                  ; *** Change this if necessary!
set input timeout proceed       ; Allow IF SUCCESS, IF FAILURE
set input echo off              ; Don't echo the modem test
output AT\13                    ; Send AT
input 2 OK                      ; Modem should say "OK"
if fail errstp {Turn on or connect your modem!}
set count 5                     ; Set up dialing loop
set input echo on               ; From now on, show what happens
echo Dialing \%1, wait...
goto dial                       ; 1st time, skip Redialing message
:REDIAL
echo Redialing...               ; Message for redialing.
:DIAL
output ATD\%1\13                ; Dial the number.
set alarm 60                    ; Detect keyboard interruptions
input 55 \10                    ; Wait for linefeed echo and for
input 5 \10                     ; linefeed after result message
if success goto gotmsg          ; Got a result message
if alarm errstp {No response from modem.} ; No response in 60 secs
hangup                          ; User interrupted from keyboard
goto again                      ; So try again right away
:GOTMSG
reinput 2 CONNECT               ; Got message, was it CONNECT?
if success goto speed           ; Yes, go check the speed
reinput 2 BUSY                  ; Not CONNECT, was it BUSY?
if failure errstp {No dialtone or no answer.}
Echo Busy...                    ; Phone was busy, give message
hangup                          ; Hang up
pause 60                        ; Wait one minute
:AGAIN
if count goto redial            ; Then go redial

errstp {It never answers!  I give up.} ; Too many tries

:SPEED                          ; Connected!
echo \7                         ; Celebrate with a beep
reinput 1 1200                  ; Was message CONNECT 1200?
if success set speed 1200       ; Yes, change the speed
```

This script is just about as smart as you are! You can use it to type any DIAL command you like at the Kermit prompt:

```
MS-Kermit>dial 555-1212         (Local call)
MS-Kermit>dial 1-800-765-4321   (Long distance)
MS-Kermit>dial 3210             (Local extension)
MS-Kermit>dial T7654321         (Force Tone dialing)
MS-Kermit>dial P7654321         (Force Pulse dialing)
MS-Kermit>dial {555 1212}       (Use braces to include spaces)
```

Now that you've seen how to construct a powerful, intelligent, and robust DIAL command for a Hayes modem, you will be able to make DIAL commands for any other kind of

modem, as well as for data PBXs, terminal servers, and any other kind of communication device that connects you to another system. And you can use the same techniques to construct scripts for logging in to any kind of computer or service.

Constructing a Dialing Directory

So you don't have to remember the phone numbers for the computers and services that you call, let's construct a dialing directory, an ordinary text file that you can create with EDLIN or your favorite word processor (remember to save it "text-only"). The dialing directory file looks like this:

```
SprintNet  555-8100  2400  mark
Tymnet     555-4700  1200  even
CompuServe 555-1423  2400  none
Office     555-1234  2400  even
Home       555-9876  2400  none
```

(These are not real telephone numbers.) Each line consists of four "fields": a name, the associated phone number, the dialing speed, and the parity setting to use. The fields are separated from one another by one or more spaces. If you want a field to contain spaces, enclose it in curly braces:

```
{Dow Jones}  {212 555 4321}  2400  none
```

If you omit the speed and parity, your current settings are used. We will call this file `DIALUPS.TXT` and keep it in the `\KERMIT` directory along with the other files that were installed there so Kermit can always find it no matter what your current directory is. In our examples, we assume that the `\KERMIT` directory is on the C disk.

Once you have created your dialing directory file, you can DIAL by name rather than by number. But first we need to write the macros that use the dialing directory file, and for this we need to learn a few new concepts and commands.

Permanent Variables

A *variable* is a symbol that stands for something else, like *x* and *y* in algebra. Macro arguments like the DIAL macro's `\%1` argument are *temporary variables* that exist only within the macro and in any command files that the macro might TAKE.

Kermit also has *permanent variables* that you can refer to anywhere: in interactive commands, in command files, and in any macro. Their names look like macro arguments, but spelled with the letters `a`–`z` rather than the digits `0`–`9` (upper and lowercase letters are equivalent). You can create permanent variables with the DEFINE command:

```
MS-Kermit>define \%a Max
MS-Kermit>echo My name is \%a.
My name is Max.
```

You can also use macros as variables, but you need special notation to refer to them: `\m(xxx)` (where `xxx` is the macro name), meaning "insert definition of macro `xxx` here:"

```
MS-Kermit>define number 7654321
MS-Kermit>echo The number is \m(number).
The number is 7654321.
MS-Kermit>dial \m(number)
```

As you can see, permanent variables are handy for remembering things from one macro call to another. Long variable names improve the clarity of your script programs by letting you name your variables according to their purposes.

Defining versus Assigning

The DEFINE command takes your definition literally. If it includes any variables, their *names* go into the definition and they are evaluated each time the macro is executed:

```
MS-Kermit>define greet echo \%1\44 \%a!
MS-Kermit>greet Hello
Hello, Max!
MS-Kermit>define \%a Andy!
MS-Kermit>greet Goodbye
Goodbye, Andy!
```

But there is more than one way to create a variable. The ASSIGN command does what DEFINE does, except it replaces all variables in the definition by their current values at the time the assignment is made, rather than just copying their names:

ASSIGN *name definition*
: Replace all variables in the *definition* by their current values. The resulting text becomes the value of the variable (or macro) called *name*.

To understand the difference between ASSIGN and DEFINE, try typing these commands (the SHOW MACRO *name* command displays the current definition of the macro or variable with the given name):

```
MS-Kermit>define \%a original   (DEFINE a variable \%a)
MS-Kermit>define \%b \%a        (DEFINE \%b to be \%a)
MS-Kermit>show macro \%b        (Look at \%b's definition)
 \%B = \%a                      (DEFINE copies the variable name)
MS-Kermit>assign \%c \%a        (ASSIGN \%a's value to \%c)
MS-Kermit>show macro \%c        (Look at \%c's definition)
 \%C = original                 (ASSIGN uses the variable's value)
MS-Kermit>define \%a new        (Now change \%a)
MS-Kermit>echo \%b              (See what happens to \%b)
new                             (It's the new value of \%a)
MS-Kermit>echo \%c              (See what happens to \%c)
original                        (It's the original value of \%a)
```

Here is a more useful example. Suppose you used the DIAL macro to dial a long complicated phone number, but the line was busy. Wouldn't it be nice if Kermit remembered the

phone number so you wouldn't have to type it again? It would be like having a REDIAL button on your telephone! With the ASSIGN command, it's easy:

```
define dial assign number \%1, take hayes.scr
define redial dial \m(number)
```

Now when you type a DIAL command, the phone number is assigned to the NUMBER macro so Kermit remembers it after DIAL finished. If you want to dial the same number again, just type REDIAL. If the DEFINE command had been used here instead of ASSIGN, the value of `\m(number)` would be literally "`\%1`", which would not work; inside of the REDIAL macro, `\%1` stands for REDIAL's first argument, not DIAL's argument.

Comparing Things

Until now, we have relied on the user to type the right number of arguments to each macro. What happens when DIAL is typed alone, with no arguments? Try it and see (you won't like it). A good macro should check whether it was invoked with the right number of arguments and take corrective action (such as supplying a default value or printing an informative error message) when necessary. More tools are needed:

IF EQUAL *word1 word2 command*

The IF EQUAL command compares two words to see if they are the same (equal). If they are, the *command* is executed, for example:

```
if equal \%1 hello echo You typed hello!
```

You can also compare words "lexically" (alphabetically) using IF LLT (lexically less than) and LGT (lexically greater than). Alphabetic case in these character string comparisons is treated according to SET INPUT CASE IGNORE or OBSERVE.

IF `=` *number number command*

The IF = command compares two numbers, or variables whose values are numeric, to see if they are numerically equal. If the numbers are equal, the *command* is executed. Variables can be macro arguments, permanent variables, or built-in named variables like `\v(argc)` (explained later in this chapter). In place of the = sign, you can use `>` for "greater than" and `<` for "less than," and you can prefix any of these with the word NOT to reverse their meaning, for example, "`NOT <`" means "not less than," which is the same as "greater than or equal to." Remember: Use EQUAL, LLT, and LGT for comparing character strings. Use =, <, and > for comparing numbers.

IF DEFINED *name command*

The IF DEFINED command succeeds and executes the *command* if the named variable or macro is defined and fails if it isn't defined or if its definition is empty. Example:

```
if defined \%a echo Your name is \%a.
```

You can reverse the sense of this statement (like any IF statement) using NOT:

```
if not defined \%a echo I don't know your name.
```

Reading and Writing PC Files

To make use of our dialing directory, we need the ability to read the entries from it so Kermit can compare them with the name that is given to the DIAL command. MS-DOS Kermit gives you a simple way to read, create, and write files on the PC. Before you can use or create a file, you have to open it, and when you are finished with the file, you must close it. A file can be opened for reading or writing, but not both. However, you can have one file opened for reading at the same time that you have another file opened for writing. The commands are OPEN, READ, WRITE, and CLOSE.

OPEN READ *filename*

This command opens the named file on your PC for reading by the READ command. Only one READ file can be open at once. If the file cannot be found or can't be opened for some other reason, or if another READ file is already open, the OPEN command fails. Example:

```
open read dialups.txt
if error fatal {Can't open DIALUPS.TXT}
```

FATAL is a "fatal error" macro. It just prints its argument (an error message) and returns to the MS-Kermit prompt:

```
define fatal echo \%1, stop
```

You can also use the OPEN command to create files or to add information to existing files:

OPEN WRITE *filename*

This command creates a new PC file called *filename* so you can write text into it with the WRITE FILE command. If a file of the same name already exists, it is destroyed. If the file cannot be created, or if another WRITE or APPEND file is already open, the OPEN WRITE command fails. Example:

```
open write report.log
if error fatal {Can't create REPORT.LOG}
```

OPEN APPEND *filename*

This command is just like OPEN WRITE, except that if the file already exists, it is not destroyed. Instead, new material is written to the end of it, after the original information. If the file does not exist a new file is created.

```
open append report.log
if error fatal {Can't append to REPORT.LOG}
```

The WRITE or APPEND file can also be a DOS device, such as the printer device, PRN. Only one file may be open for writing at a time. In total, one file may be open for reading and one file for writing or appending. This lets you write script programs that read text from one file and write or append text to another file.

Once a file is open, use these commands to read from it or write to it and to close it:

READ *variable-name*

This command reads a line of text from the file most recently opened by an OPEN READ command into the specified macro argument (if the READ command is issued from a macro) or permanent variable. The line termination characters (carriage return and linefeed) are discarded. The command fails if a READ file is not open, or if there are no more lines left in the file. Example:

```
read \%a                        ; Read a line into variable \%a.
if failure fatal {End of file}  ; Failed, end of file.
```

WRITE FILE *text*

This command writes the given text, which may contain any mixture of ordinary characters, backslash codes, and variables, to the file most recently opened by an OPEN WRITE or OPEN APPEND command. Line terminators are not supplied by this command, so if you need them you must include the appropriate backslash codes. The WRITE FILE command fails if a WRITE or APPEND file is not open, or if there is any kind of error in the DOS write operation, such as a disk filling up. Example:

```
write file This is a line of text\13\10
if failure fatal {Error writing file!}
```

CLOSE READ-FILE

This command closes the currently open READ file, if any. You can't issue any further OPEN READ commands until you close the READ file.

CLOSE WRITE-FILE

This closes the currently open WRITE or APPEND file, if any. This forces DOS to make sure all your WRITE commands have been completed and to update the size, date, and time in the file's disk directory entry (if your WRITE file was on a disk).

To illustrate, here is a short program that displays and copies a file:

```
open read file.old              ; Open the file to be copied.
if failure fatal {Can't open FILE.OLD}
open write file.new             ; Open new copy of file.
if failure fatal {Can't create FILE.NEW}
:loop
read \%a                        ; Read a line of text into \%a.
if failure goto eof             ; No more lines left?
echo \%a                        ; Got a line, echo it on screen.
write file \%a\13\10            ; Write line to new file.
goto loop                       ; Go back and read next line.
:eof                            ; Come here at end of file.
close read                      ; Close read file.
close write                     ; Close write file.
```

Look It Up and Dial!

Now let's use our new tools—permanent variables, the ASSIGN command, comparisons, the FATAL macro, and the OPEN, READ, and CLOSE commands—to construct a new DIAL macro, complete with dialing directory and automatic recall of the last number dialed. Kermit's built-in `\v(argc)` variable tells the number of words in the macro invocation (the argument count). For example, if you type `dial 1234`, `\v(argc)` is 2. We use this feature to detect missing (or extra) macro arguments:

```
define dial -
  if = \v(argc) 2 assign number \%1,-
  if < \v(argc) 2 if not def number fatal {Dial what?},-
  assign \%1 \m(number),-
  take hayes.scr
```

If the user types just `dial` and the macro `number` is not yet defined, the DIAL macro says, "Dial what?" and quits, using the FATAL macro. But if a number *was* previously dialed, the macro will find it in `\m(number)` and use it. If a new number is specified, it replaces the old one as the value of `\m(number)`. Finally, the value of `\m(number)` is assigned to `\%1` because that is where the `HAYES.SCR` command file expects to find it, and then the `HAYES.SCR` file is executed with the TAKE command.

Now we know what the user wants to dial. If the user types a name instead of a number, we can look it up in our dialing directory. We will write a LOOKUP macro to read the dialing directory file, `C:\KERMIT\DIALUPS.TXT`. To look up a number, we read lines from the file until we find one that starts with the name that was given to the DIAL command. We use a special macro to split the items (name, number, speed, and parity) from each line and assign them to separate permanent variables. We'll call it SPLIT:

```
define split assign \%x \%1, assign \%y \%2,-
  assign \%s \%3, assign \%p \%4
```

We use the SPLIT macro like this:

```
read \%9                        ; Read line into temporary variable
split \%9                       ; Assign each word to a variable
```

The READ command reads a line such as:

```
SprintNet  555-8100  2400  mark
```

into the variable `\%9`, and the SPLIT macro puts the name (first word of the line) into the variable `\%x`, the phone number into `\%y`, the speed (if any) into `\%s`, and the parity (if any) into `\%p`.

Now we can write the LOOKUP macro. It opens the dialing directory file, reads each line and splits it up, and compares the name with what the user typed. If they match, LOOKUP

assigns the telephone number to \m(number), sets the speed and parity as specified, and closes the dialing directory file.

```
def lookup -
  open read c:\kermit\dialups.txt,-
  if fail fatal {Can't open dialing directory},-
:loop,-
  read \%9,-
  if fail goto eof,-
  split \%9,-
  if not eq \%x \%1 goto loop,-
  assign number \%y,-
  if def \%s set speed \%s,-
  if def \%p set parity \%p,-
:eof,-
  close read
```

Notice, once again, how commas are used instead of carriage returns to separate Kermit commands within a macro definition, and how the definition can be continued onto multiple lines by using hyphens. And note how the last line of each macro definition does *not* have a comma or hyphen. Also note that continued lines do not (and must not) have trailing comments. If they did, the rest of the macro definition would be ignored.

The DIAL macro uses LOOKUP like this:

```
lookup \m(number)
```

This assigns the value of \m(number) (say, "compuserve") to LOOKUP's first positional macro argument, \%1. If LOOKUP finds this word in the directory, it redefines \m(number) to be the associated phone number, in this case 555-1423, and sets the appropriate speed and parity. If it doesn't find the word, it does not change the value of \m(number). This way, you can still DIAL phone numbers that aren't in your dialing directory. So let's add the LOOKUP macro to our new DIAL command:

```
def dial if = \v(argc) 2 assign number \%1,-
  if < \v(argc) 2 if not def number fatal {Dial what?},-
  if > \v(argc) 2 fatal {No spaces please.},-
  lookup \m(number),-
  assign \%1 \m(number),-
  take hayes.scr
```

Before you can use the DIAL, FATAL, SPLIT, and LOOKUP macros, Kermit must execute DEFINE commands for them. You will find these macros already defined for you in the MSKERMIT.INI file supplied on your MS-DOS Kermit 3.11 distribution diskette, so if you have installed Kermit properly, their definitions will always be available.

With these definitions in place, and your DIALUPS.TXT file stored in the \KERMIT directory, you can type commands like these at any time:

```
MS-Kermit>dial 7654321          Dial a phone number
MS-Kermit>dial compuserve       Use the dialing directory
MS-Kermit>dial                  Redial most recent number
```

If you would like to add a LIST command to display your dialing directory, it's easy:

```
define list run more < c:\kermit\dialups.txt
```

or if you want to be able to display specific entries:

```
define list -
 if < \v(argc) 2 run more < c:\kermit\dialups.txt,-
 if > \v(argc) 1 run find "\%1" c:\kermit\dialups.txt | more
```

The DOS FIND command displays lines containing the given quoted text (alphabetic case matters) in the given file. Here it looks in the dialing directory file for whatever word you type after LIST:

```
MS-Kermit>list                  (No argument, list them all)
SprintNet  555-8100  2400  mark
Tymnet     555-4700  1200  even
CompuServe 555-1423  2400  none
Office     555-1234  2400  even
Home       555-9876  2400  none
MS-Kermit>list Tymnet           (Argument given, use FIND)
---------- c:\kermit\dialups.txt
Tymnet     555-4700  1200  even
MS-Kermit>
```

Now you are ready to use and inspect your dialing directory whenever Kermit is active. All your phone numbers are now committed to the computer's memory, so you can erase them from your brain to make room for more new Kermit commands.

Automated File Transfer

If you've come this far, you're probably asking, "Why not go one step further and automate my entire session with the remote computer?" Good idea. Let's do it.

You've got all the tools you need, but here's a safety tip before we proceed. To have a fully automated session with a remote host or service, you must get Kermit to send it your access codes. But are you going to put your secret password in your script file?

Let's hope not! Anyone who has access to the messy pile of diskettes on your desk could find your password and use it to delete all of your files on the host computer, run up a big bill on your credit card, or worse! So let's design our procedure so that your password does not need to be stored on a disk.

We can put our procedure together from the tools we've created already. But we begin with something new: We prompt the user for the login access codes, so these codes can be kept in Kermit's memory temporarily rather than on the disk permanently—a good security precaution. The command is:

ASK *or* **ASKQ** *variable-name prompt*

Prints the prompt. Whatever the user types in response is assigned to the variable, for example:

```
MS-Kermit>ask \%p Password:
Password:secret
MS-Kermit>
```

If you use ASK, characters are echoed on the screen as the user types them. With ASKQ the user's keystrokes do not echo, a useful safety feature when typing passwords:

```
MS-Kermit>askq \%p Password:
Password:______
MS-Kermit>
```

HINT: If you want your prompt to have trailing spaces, use braces:

```
MS-Kermit>ask \%p {Password: }
Password: secret
MS-Kermit>
```

We begin our automated procedure by collecting the information we need from the user, in prompting style:

```
cls
echo Welcome to the Kermit automat.
echo Do as I say and we'll get along fine.
echo
:number
ask \%n What number should I dial?\32
if not def \%n goto number
assign number \%n
:userid
ask \%u What is your user ID?\32
if not def \%u goto userid
:password
askq \%p What is your password?\32
if not def \%p goto password
echo
echo OK - here we go...
```

Try these yourself. Notice how you are required to enter each item. If you just press the Enter key, the program issues a prompt for the same item again. Once the phone number, username, and password are collected, the script can dial the phone number:

```
dial \m(number)                 ; Dial the number
```

If dialing is unsuccessful, the script stops automatically (see HAYES.SCR).

Now let's adapt the VAX/VMS login from page 161 to use the username and password variables we have just collected:

```
set input timeout proceed
output \13                      ; Send a carriage return
input 10 Username:              ; Look for the VMS Username prompt
if fail errstp {No Username prompt from host}
output \%u\13                   ; Send the username
input 10 Password:              ; Get the password prompt
if fail errstp {No Password prompt from host}
output \%p\13                   ; Send the password
define \%p                      ; Erase the password from memory
input 40 $\32                   ; Get the VMS prompt "$ "
if fail errstp {Login failed}
```

If the script program gets this far, it has logged in to the VAX computer. Now you can have it transfer some files back and forth. Let's assume that your PC is in a branch office of your company, that every night you must send a log (`TODAY.LOG`) of the day's activities to corporate headquarters, and that you also have to fetch a list of new orders to be filled (`ORDERS.LST`). The easiest way to do this is to use the Kermit server on the VAX:

```
output kermit\13                ; Start Kermit on host
input 10 C-Kermit>              ; Wait for its prompt
if fail errstp {Can't run Kermit on host}
output server\13                ; Put it in server mode
input 10 reconnect.             ; Get end of server's greeting
if fail errstp {Can't enter server mode on host}
send today.log                  ; Send today's log
if fail errstp {Send TODAY.LOG failed}
get orders.lst                  ; Get new orders
if fail errstp {Get ORDERS.LST failed}
logout                          ; Done, send server away
if exist yesterda.log del yesterda.log
run copy today.log yesterda.log
echo All's well that ends well.
```

If these script fragments are collected into a file called, say, `NIGHTLY.SCR`, the simple command TAKE NIGHTLY from the `MS-Kermit>` prompt sets the entire chain of events into motion. This shows how scripts can be more than a simple convenience: They allow you to automate procedures for unskilled people. A copy of `NIGHTLY.SCR` could be installed on the PC in each of your branch offices, and every evening the last person to leave could start Kermit and type `take nightly.scr` at the `MS-Kermit>` prompt.

These employees don't have to remember numerous commands or learn what they really don't care to know. All they have to do is start their PC, run Kermit, and type one magic command. If that was too hard, you could further automate the process for them by writ-

ing a DOS batch file,reducing the process to one single-character DOS command, so that they wouldn't have to know a thing about Kermit:

```
kermit take nightly.scr
```

Call this file `X.BAT`, and people only have to type `X` (and Enter) to run the whole procedure. You are not only a hero but probably much wealthier. Remember your friends (and the one who taught you this trick).

Wait, here's one more! Wouldn't it be better if all this uploading took place late at night, when the phone rates were lowest and the corporate mainframe the least busy? Simple: In `NIGHTLY.SCR`, just before the DIAL command, include a line like:

```
pause 23:59:59
```

This will make Kermit wait until midnight and then dial. No humans need to be present. See, saved you another few bucks!

Advanced Topics

You've seen how to use Kermit commands interactively, in command files, and in macros, and how to use variables to substitute values, such as a telephone number to be dialed, into Kermit commands. Kermit also has other kinds of variables that let you get at information that only Kermit—or the computer itself—knows. Let's take a look at them.

Built-In Variables

Kermit's *built-in* variables have names that look like `\v(`*name*`)`, for example `\v(date)`. You can use these names anywhere in a Kermit command, but you can't change their values. Kermit's built-in variables are:

`\v(argc)`
: The number of arguments passed to the current macro.

```
MS-Kermit>define howmany echo \v(argc) words.
MS-Kermit>howmany is this
3 words.
```

`\v(count)`
: The current value of the SET COUNT variable. Unlike the COUNT variable, it can be used in any context and does not have one subtracted from it every time you use it:

```
set count 5
:loop
echo \v(count)
if count goto loop
echo Zero!
```

`\v(directory)`
: Current disk and directory, for example `C:\KERMIT`.

`\v(errorlevel)`
Current value of the SET ERRORLEVEL variable.

`\v(date)`
The current date, expressed in national format. For example, Valentine's Day 1991 would be 02/14/1991 in the USA, 14/02/1991 in Europe, and 1991/02/14 in Japan:

```
MS-Kermit>echo Today is \v(date).
Today is 02/14/1991.
MS-Kermit>
```

`\v(ndate)`
The current date, expressed in universal numeric format. Valentine's Day 1991 is 19910214. This variable is suitable for sorting or for constructing DOS filenames:

```
MS-Kermit>receive \v(ndate).log
```

`\v(keyboard)`
88, 101, or 250 to denote the keyboard type: 88 is the original, small PC keyboard; 101 is the enhanced keyboard with F11-F12, separate editing and arrow keypads. 250 means that a DEC LK250 keyboard is installed, selected (SET KEY LK250), and an external driver is present. Use `\v(keyboard)` to construct key-setting files that can be used without modification on PCs with different kinds of keyboards, for example:

```
goto \v(keyboard)
:88
set key commands for 88-key model
goto done
:101
set key commands for 101-key model
goto done
:250
set key commands for LK250
:done
```

`\v(platform)`
The value is IBM-PC for IBM PCs, PS/2s, and compatibles.`\v(platform)` has appropriate values for MS-DOS Kermit on non-IBM compatibles like the HP-150, DEC Rainbow, etc.

`\v(program)`
The value is always MS-DOS_KERMIT. Use this for constructing Kermit script programs that can be used with C-Kermit (5A and later) as well as MS-DOS Kermit. It allows the script program to test which Kermit program is active:

```
if equal \v(program) MS-DOS_KERMIT set port 1
if equal \v(program) C-KERMIT set line /dev/cua1
```

`\v(speed)`
The transmission speed of the current communication port, in bits per second if known; otherwise "unknown".

`\v(status)`
0 if the previous command succeeded, nonzero if it failed. Use this variable for remembering the success or failure of a command until a later time as in this example, which deletes the file `FOO.TXT` only if both it and another file were sent successfully:

```
send foo.txt
assign \%a \v(status)
send bar.txt
if success if = 0 \%a delete foo.txt
```

`\v(system)`
The value is always MS-DOS.

`\v(time)`
The current time in 24-hour `hh:mm:ss` format, for example `20:52:42`.

`\v(version)`
The numeric MS-DOS Kermit program version number. For example, `\v(version)` is 311 for version 3.11. Use this variable to test for version number dependencies in a command file:

```
if < \v(version) 311 goto common
put commands here that didn't exist before version 3.11
:common
put commands here that work in all versions
```

You can see a list of MS-DOS Kermit's built-in variables and their values by giving the SHOW VARIABLES command.

Environment Variables

Besides its own permanent variables, MS-DOS Kermit gives you access to DOS environment variables. A DOS environment variable is created when you give the command SET *name=value* at the DOS prompt or in a batch file (like `AUTOEXEC.BAT`), for example:

```
set path=d:\;c:\;c:\local;c:\kermit
```

You can find out what your environment variables are by typing SET (by itself) at the DOS prompt:

```
C:\>set
COMSPEC=D:COMMAND.COM
KERMIT=ROLLBACK 40
PATH=D:\;C:\;C:\LOCAL;C:\KERMIT
PROMPT=$P$G
```

You can refer to a DOS environment variable within Kermit using the notation `\$(`*name*`)` (note: `$` instead of `v`), where *name* is the name of environment variable, for example:

```
Kermit-MS>echo My PATH is \$(path).
My PATH is D:\;C:\;C:\LOCAL;C:\KERMIT.
```

Alphabetic case in the variable name does not matter: `PATH` is the same as `path`.

You can use environment variables to let a single Kermit command file or macro take advantage of how different people's PCs are set up. For example, suppose at a certain company, everybody's AUTOEXEC.BAT file contained commands like these:

```
set myname=Julie
set mydept=Accounting
set mybirthday=03/04/92
set home=C:\JULIE
```

This lets everybody have the same MSKERMIT.INI file and still get personalized service:

```
ask \%a Hello, \$(myname), how's life in \$(mydept) today?
if eq \%a terrible echo I'm sorry to hear that.
if eq "\$(mybirthday)" "\v(date)" echo Happy Birthday, \v(myname)!
```

The Transaction Log

Unattended operations should leave a record of what they have done and what they failed to do. Kermit's method for doing this is called the transaction log. The command that activates transaction logging is (you guessed it):

LOG TRANSACTIONS *filename*

This creates a file called TRANSACT.LOG, that contains a record of all Kermit's file transfer operations. You can inspect this file in the morning to see whether all went as planned. You can add your own entries to the transaction log with the WRITE TRANSACTION command, for example:

```
write transact \v(date) Nightly run from \v(dir) at \v(time)\13\10
```

This produces a single line like:

```
02-08-1991 Nightly run from C:\OOFA at 23:59:59
```

The WRITE command is also available for the screen (WRITE SCREEN), the session log (WRITE SESSION), and the packet log (WRITE PACKET), for example:

```
write screen It's \v(time) - Do you know where your children are?
```

Putting It All Together

Let's look at the complete script resulting from all our new knowledge. We've added transaction logging, a little extra bulletproofing, plus creative use of Kermit's built-in variables. Here is NIGHTLY.SCR.

```
:LOG
def tlog write trans \v(time): \%1
log trans \v(ndate).log         ; Open the transaction log
if exist today.log goto ok      ; Make sure TODAY.LOG exists
tlog {TODAY.LOG not found.}
echo TODAY.LOG not found.
stop                            ; Bad news...
```

```
:OK                             ; TODAY.LOG exists
cls                             ; Clear screen, begin dialog.
echo Welcome to the Kermit automat.
echo Do as I say and we'll get along fine.
echo

:NUMBER                         ; Prompt for phone number
ask \%n What number should I dial?\32
if not def \%n goto number
assign number \%n

:USERID                         ; Ask for VAX user ID
ask \%u What is your VAX user ID?\32
if not def \%u goto userid

:PASSWORD                       ; Ask Quietly for Password
askq \%p What is the password for \%u?\32
if not def \%p goto password

echo
echo OK - Sleeping till midnight...
pause 23:59:59                  ; Wait until midnight
set input timeout proceed       ; Use IF SUCC/FAIL
dial \m(number)                 ; Dial the phone number
def errstp2 echo Error: \%1, hangup,-
 tlog {\%1}, close trans, stop
tlog {Nightly Procedure Begins...}
output \13                      ; Send carriage return to the VAX
input 10 Username:              ; Look for Username prompt
if fail errstp2 {No Username prompt from host}
output \%u\13                   ; Send the username
input 10 Password:              ; Get password prompt
if fail errstp2 {No Password prompt from host}
output \%p\13                   ; Send the password
define \%p                      ; Erase password from memory
input 30 $\32                   ; Get VMS prompt "$ "
if fail errstp2 {Login failed!}

output run kermit\13            ; Start Kermit on host
input 10 C-Kermit>              ; Wait for its prompt
if fail errstp2 {Can't run Kermit on host}
output server\13                ; Put it in server mode
input 10 reconnect.             ; Get end of server's greeting
if fail errstp2 {Can't enter server mode on host}

set display quiet               ; No file transfer screen
send today.log                  ; Send today's log
if fail errstp2 {Send TODAY.LOG failed}

get orders.lst                  ; Get new orders
if fail errstp2 {Get ORDERS.LST failed}

logout                          ; Done, send server away
close trans                     ; Close transaction log
echo All's well that ends.
```

Predefined Macro Names

Several macro names are special to MS-DOS Kermit. These macros can be defined by you to do anything you like, and Kermit executes them automatically (if you have defined them) under certain conditions. Of course, you also can invoke these macros in all the normal ways: typing their names at the `MS-Kermit>` command prompt, assigning them to a key, and so forth. MS-DOS Kermit's predefined macro names are ON_EXIT, TERMINALS, TERMINALR, and PRODUCT.

The ON_EXIT Macro

Just as there are things you want Kermit to do every time it starts, there might be things you want Kermit to do every time it exits: put the screen back in a certain mode, turn off the internal modem, etc. To do this, define macro called ON_EXIT in your `MSKERMIT.INI` file and put any commands you want into its definition:

```
define on_exit mode modem off, -
  run sa black on green, -
  echo bye bye
```

Kermit executes this macro (if it exists) automatically whenever you give an EXIT or QUIT command, or if it exits for any other normal reason.

The TERMINALS and TERMINALR Macros

Two macros named TERMINALR and TERMINALS are invoked automatically—if you have created definitions for them—during CONNECT mode when the remote host sends the special escape sequences `ESC [ ? 34 l` (the final character is lowercase letter L) and `ESC [ ? 34 h`, respectively. See Appendix II for an explanation of escape sequences.

To illustrate the power of TERMINALS and TERMINALR, let's see how we can set things up so that all file transfer operations can be controlled completely by the commands that you give to the remote Kermit. Assume the remote host is running C-Kermit 5A or later. Our first step is to define the following C-Kermit macros, either at the C-Kermit prompt or in a C-Kermit command or initialization file:

```
define pcsend echo \27[\{63}34l, send \%1 \%2
define pcget echo \27[\{63}34h, get \%1, finish
```

The PCSEND macro sends the escape sequence that invokes MS-DOS Kermit's TERMINALR macro, and then sends the given file (`\%1`) under whatever alternate name (`\%2`) we tell it, if any. For example:

```
C-Kermit>pcsend oofa.new
C-Kermit>pcsend oofa.new pseudo.nym
```

The PCGET macro sends MS-DOS Kermit's TERMINALS sequence, issues a GET command for the specified file, and then sends a FINISH command:

```
C-Kermit>pcget oofa.old
```

Have you guessed yet what the definitions of TERMINALR and TERMINALS should be? Here they are:

```
define terminalr receive, connect
define terminals server, connect
```

With these definitions in place, you never have to escape back, reconnect, or even press an F-key to get MS-DOS Kermit into file transfer mode. Just use PCSEND at the remote C-Kermit prompt instead of SEND, PCGET instead of RECEIVE, and MS-DOS Kermit follows along obediently. Try it!

The PRODUCT Macro

A macro called PRODUCT, if you have created a definition for it, is invoked when the host sends the escape sequence `ESC [ Pn;...Pn; ~` (see Table II-12). The `Pn`'s are numbers, and there can be up to nine of them. For example:

```
ESC [ 2;4;6;8;10 ~
```

These numbers become the arguments of the PRODUCT macro. In the example the arguments are `\%1` = 2, `\%2` = 4, `\%3` = 6, `\%4` = 8, and `\%5` = 10. The arguments `\%5` through `\%9` are undefined (empty). You can create a PRODUCT macro that uses these arguments for anything you like.

But what is the PRODUCT macro really for? It is intended to be used by host-resident software packages that need to configure Kermit in particular ways. The first argument is supposed to be a code that identifies the package vendor, the second is a vendor-specific product code, and the remaining arguments are product-specific codes.

Let's say that all the software vendors have agreed to assign themselves unique vendor codes. So now we have code 0 for ABC Corporation, code 1 for XYZ Corporation, and so on. Let's say CMG Corporation markets a package called VAX-AGIDA that wants the PC keyboard to be set up in a certain elaborate way, and VAX-AGIDA customers have been given an MS-DOS Kermit command file, `VAXAGIDA.INI`, containing lots of SET KEY commands for this purpose. The vendor code for CMG Corporation is 35, the product code for VAX-AGIDA is 0, and argument number 3 is 0 to set up the keyboard for VAX-AGIDA and 1 to restore Kermit's default keyboard configuration.

Here's how it would work. You start Kermit on your PC, connect to the host, and start up the VAX-AGIDA application there. It sends the escape sequence:

```
ESC [ 35;0;0 ~
```

which says, "This is a message from CMG Corporation; VAX-AGIDA is starting." When you exit from VAX-AGIDA on the host, it sends:

```
ESC [ 35;0;1 ~
```

meaning "Goodbye from VAX-AGIDA: As you were!"

On the MS-DOS Kermit end, you must have created a definition for the PRODUCT macro, such as:

```
def product if = \%1 35 if = \%2 0 if = \%3 0 take vaxagida.ini,-
  if = \%1 35 if = \%2 0 if = \%3 1 set key clear, connect
```

This discussion has been purely hypothetical. Vendors of host-resident products that expect their software to be used by PC-resident terminal emulators like Kermit are encouraged to get together and adopt a mechanism like this. Meanwhile, creative Kermit users will no doubt find their own uses for it!

Command Files versus Macros

There is practically no limit to what you can do with MS-DOS Kermit's command files, macros, and scripts. The techniques illustrated in this chapter can be used to construct procedures to carry on virtually any kind of interaction with a modem or a remote host computer or dialup service. The complexity and length of a command file are limited only by your imagination and the size of your disk. Macros, however, have some limitations that you should keep in mind:

- The maximum length for a macro definition is 1000 characters. Sometimes you have to abbreviate a lot to make the definition fit, which affects clarity.
- Comments are allowed only at the end of a macro definition, not on each line. Again, clarity suffers.
- The total memory available for macro names is about 1000 characters, so if you need to define many macros, keep their names short.
- Memory for macro definitions is allocated dynamically from DOS, so space is restricted to the amount of free memory that is not taken up by the Kermit program itself and any other programs that you have loaded.
- You must define a macro before you can use it. That is, the DEFINE command for the macro must be executed before you can use the macro.

It is a good idea to collect the definitions of frequently used macros into a file that is TAKEn automatically each time you run MS-DOS Kermit. Your MS-DOS Kermit initialization file, `MSKERMIT.INI`, is the logical place.

Chapter 15

Use of MS-DOS Kermit by People with Visual, Auditory, or Physical Challenges

Unlike most other communication software packages for the PC, MS-DOS Kermit includes features that allow it to be used by people who are partially sighted, blind, deaf, or who do not have full use of their hands.

FEATURES FOR THE VISUALLY CHALLENGED

Partially sighted people might be able to read the PC's large-print 40-column display better than its normal 80-column display. For large print, use the DOS command:

```
C>mode 40
```

before starting the Kermit program. Kermit senses the current screen width and behaves appropriately. At the MS-DOS Kermit prompt, messages that are longer than 40 characters wrap to the next line. During terminal emulation, you should inform your host computer that your

screen width is 40 rather than 80 so screen-oriented applications can adjust to your screen size. Two Kermit commands are also useful during terminal emulation:

SET TERMINAL WRAP ON

This ensures that lines of text that are longer than the current screen width are not lost.

SET TERMINAL CURSOR BLOCK

This makes a very big cursor that is much easier to see than Kermit's normal underline cursor.

Accessing the Port

People who cannot see at all are able to use PCs by installing speech synthesis or Braille devices, which speak or form the words that appear on the screen. For these devices to be useful, text must be displayed on the screen in a linear fashion. Speech synthesis and Braille printing do not work well with applications that use pop-up menus, graphics, colors, or other fancy displays, or that update random fields on the screen as Kermit does during VT terminal emulation or in its regular file transfer display.

Kermit can be instructed to write all material to the screen in a way that Braille and speech devices can interpret sensibly. The commands are:

SET TERMINAL TYPE NONE

During terminal emulation, Kermit ignores all screen formatting and cursor positioning commands from the host, and displays all arriving characters in linear fashion. For best results, you should also tell the host that you are using a hardcopy (printing) terminal so it will not send unwanted screen formatting commands. Refer to Chapter 8 about setting your terminal type on the host.

SET TERMINAL MARGIN-BELL ON

This command can be used to have Kermit ring its bell when the cursor approaches the right screen margin, similar to the margin bell on a typewriter.

SET FILE DISPLAY SERIAL

During file transfer, the status of the file transfer is written in linear fashion instead of as random updates to a form on the screen.

The effects of these commands can be best appreciated by trying to use Kermit without them.

In addition, the software that drives certain speech synthesizers takes over the PC's keyboard BIOS interrupt. Kermit normally uses the same method during terminal emulation, and this can cause conflicts. To resolve the conflict, use the command:

SET KEY OFF

During terminal emulation, instruct Kermit to access the keyboard using DOS rather than BIOS to give BIOS-level keyboard drivers priority in interpreting your keystrokes. If you don't know what this means, but your special devices are not working right during terminal emulation, try giving this command before you give the CONNECT command. SET KEY OFF does not interfere with your key translations, except that certain keys or combinations (*F11*, *F12*, and *Ctrl-P*, for example) will no longer be recognized by Kermit.

Finally, users of speaking devices should use the ASKQ command in scripts that prompt for passwords, so when a password is typed, it is not audible to others.

Using MS-DOS Kermit has been submitted to Computerized Books for the Blind. For further information, contact:

Computerized Books for the Blind
33 Corbin Hall
University of Montana
Missoula, MN 59812 USA
Telephone: (406) 243-5481

Features for People Who Are Deaf

MS-DOS Kermit's "visual bell" option is for those who can't hear the terminal's normal bell. The audible beeps that Kermit normally emits during terminal emulation are converted into screen flashes if you issue the command SET TERMINAL BELL VISIBLE. From then on, if the host sends beeps to get your attention, you will be able to see them.

Features for the Physically Challenged

People who do not have full use of their hands may find it difficult to press more than one key at a time, which is necessary for entering Shift-, Ctrl-, and Alt-key combinations. Kermit's SET KEY commandlets you assign any character or group of characters to any single key that generates a Kermit scan code. That is, if SHOW KEY prints a scan code when you press the key, you can define that key to transmit anything you like. Frequently used words or key combinations can be assigned to single keys. For example, to assign Ctrl-C to the F1 key, type:

```
MS-Kermit>set key \315 \3
```

You can assign Kermit verbs to single keys, too. For example, to put the "Escape back to Kermit prompt" function on the F10 key:

```
MS-Kermit>set key \324 \Kexit
```

These key assignments are effective during terminal emulation but not at the MS-DOS Kermit prompt or during file transfer. When typing Kermit commands, or DOS commands in general, mistakes can be corrected by using the Backspace key. You may use IBM's console driver, `ANSI.SYS`, to reprogram your keys to assign difficult key combinations to single keys that can be used at the `MS-Kermit>` or DOS prompt.

To further reduce typing, define Kermit macros to combine common sequences of commands into short commands. Macros can be invoked by pressing single keys during terminal emulation if you use SET KEY `\`*nnn* `{\K`*name*`}` to set them up (see page 152).

For general PC use, there are alternate keyboards, switches, foot treadles, joysticks, and special keyboard driver software to allow data to be input to a computer by people who don't have full use of their hands. Even very simple mechanical keylocks, which hold down the Ctrl, Alt, or Shift keys, can be a big help.

If you are using a special keyboard driver, you can ensure that Kermit does not interfere with it by giving the Kermit command SET KEY OFF.

Chapter 16

Kermit on Local Area Networks

Your serial port or internal modem is an excellent communication device. It's cheap, and it lets you establish dialup connections at whatever speeds your modem supports (typically 1200, 2400, or 9600 bits per second) and short-distance local connections at speeds up to 57,600 bps. Local Area Networks (LANs), however, allow PCs to be connected to each other and to other kinds of computers at much higher speeds, such as 1 million bits per second (Mbps), 4 Mbps, 10 Mbps, 16 Mbps, or even higher.

Many network technologies, protocols, and brand names are available for PCs and MS-DOS Kermit supports most of them. They include: AT&T StarLAN/StarGROUP, Digital Equipment Corporation DECnet (including LAT and CTERM), IBM EBIOS/LANACS, Intel OpenNET, NETBIOS, Novell NASI/NACS, Novell TELAPI, InterConnections Inc. and Novell TES, TCP/IP, 3Com BAPI, and Ungermann-Bass Net/One, plus any BIOS Interrupt 14 interceptor for other network services.

If your network includes an external session manager, you can use Kermit's `\Knethold` keyboard verb, normally assigned to the *`Alt-n`* key (hold down Alt, press lowercase n), to get the session manager's attention during terminal emulation, place network connections on hold, or switch among multiple sessions. You can also assign this function to any other key with the SET KEY command, for example to F10:

```
MS-Kermit>set key \324 \Knethold
```

The networks that provide session management include Novell NASI/NACS, Ungermann-Bass Net/One, 3Com BAPI, and InterConnections/Novell TES.

Novell Networks

To get an idea of what Kermit can do on a LAN, let's look at its support for Novell and NETBIOS-based networks. MS-DOS Kermit can operate with Novell products in at least six ways: with a file server, with an asynchronous communication server, station-to-station, with NetWare/VMS, with TES, and through a TCP/IP gateway.

File and Print Servers

Using Kermit with a file server is especially easy because the file server looks like a local PC device to Kermit; an additional set of disk drive letters and possibly extra printers. Special procedures are built into Kermit to assist printing to a network printer while Kermit is attached to a host via high-speed communications media. Use the Novell utility CAPTURE to redirect a PC printer device to a network printer.

Asynchronous Communication Servers

An asynchronous communication server (ACS) is a device on the PC network that houses serial communication ports or modems that all the PCs on the network can share, using them as if they were locally attached. On a Novell network, this is done by running the Novell NASI utility on your PC. The asynchronous server runs the matching part, named NACS. Communication programs like Kermit that understand the NASI protocol can use it to communicate with the ACS. Just give it the commands SET PORT NOVELL and CONNECT, and you're communicating with the ACS and can use its menu to select a modem or port. You can maintain multiple simultaneous connections and switch among them by pressing *Alt-n* (or whatever other key you assigned `\Knethold` to).

Station-To-Station

Station-to-station communication is also possible with Kermit through the SET PORT NETBIOS command. Any two PCs on the Novell network (or other networks supporting NETBIOS) can transfer, print, and manage files between themselves using Kermit's normal file transfer commands. This is a useful option when the file server is unavailable, when it does not provide the flexibility that you need, or when its disk is full. If you also have timesharing computer hosts on the network that support NETBIOS protocol such as UNIX hosts with AT&T StarLAN/StarGROUP (see page 193), SET PORT NETBIOS lets you CONNECT to the host, have a terminal session, and transfer files in the normal Kermit way.

Before running Kermit, run the NETBIOS program, `NETBIOS.EXE` (you can do this from your `AUTOEXEC.BAT` file). Then start Kermit and, if a node name has not already been established for your PC by some other network utility, use Kermit's SET NETBIOS-NAME command to define a unique network name for itself:

```
MS-Kermit>set netbios-name grumpy
```

Kermit automatically adds a period and an uppercase letter K to the end of name if you didn't include them in the SET NETBIOS-NAME command, or if a name was already established. This makes the name specific to Kermit-to-Kermit network sessions. Alphabetic case is significant in node names. The maximum length is 16, including the `.K`. If your PC is on a NETBIOS-based network, you should add a SET NETBIOS-NAME command to your `MSKERMIT.INI` file.

To set up your PC as a server, just type the MS-DOS Kermit command SET PORT NETBIOS, and then the SERVER command. Example:

```
MS-Kermit>set netbios-name jrd
MS-Kermit>set port netbios
 Checking if our node name is unique...
 The network is active, our name is jrd.K
MS-Kermit>server
```

The SET PORT NETBIOS command fails if your PC did not already have a NETBIOS node name when Kermit was started and you did not give it a name with the SET NETBIOS-NAME command. It also fails if you have not run `NETBIOS.EXE` before starting Kermit.

To communicate with another Kermit program on the network that has already been set up as a server, use SET PORT NETBIOS *name*, specifying the name of the server you want to talk to, and then give file transfer and REMOTE commands, as you would for any Kermit server:

```
MS-Kermit>set port net jrd.K
 Checking if our node name is unique...
 The network is active, our name is sneezy.K
MS-Kermit>remote cd \public
MS-Kermit>get mskermit.pch
```

Accessing VAX/VMS Systems

VAX/VMS computers can join Novell networks too, and PCs with appropriate software can log in to them directly over the Novell network. The InterConnections, Inc. TES program (also marketed by Novell as part of NetWare for VMS) and shell execute the low-level protocols, and MS-DOS Kermit rides on top of them. After setting up your PC as a Novell client, run the TES program, and then start Kermit and give it the commands SET PORT TES *name* and CONNECT. The target VAX must have the TES server installed, and there must either be a Novell file server somewhere on your network or your VAX must be running NetWare for VMS.

Multiple sessions are available via TES's hot-key menu to place connections on hold, or to hang up old ones. Press *`Alt-n`* (your `\Knethold` key) whenever you want to switch sessions. Using a host name of `*` shows a list of available host machines. For versions of TES prior to 2.0, use SET PORT BIOS1 instead, and use TES's own hot key (normally Alt-LeftShift-T) for session management.

TES uses NetWare IPX packets that can pass through Novell NetWare file servers and external bridges. The workstation can use any type of network board supported by NetWare, not just Ethernet. The VAX uses its DEC Ethernet board and requires that packets conform to Ethernet II Type 8137 conventions. An inexpensive external NetWare Bridge (running in an old AT machine) can join dissimilar physical networks, such as ARCNET on the workstation side bridged to Ethernet going to the VAX.

Novell TCP/IP Access

Novell File Servers prior to NetWare 386 version 3.11 understand NetWare IPX packets but not the IP packets of TCP/IP. Thus IP packets do not pass through these Novell file servers or external NetWare bridges. To let your PC establish TCP/IP connections to external TCP/IP hosts, your Novell network needs a TCP/IP gateway, such as the one marketed by Interlan. NetWare version 3.11, however, supports both TCP/IP and IPX as transport protocols, and so a special gateway is not necessary.

If your Novell network has a TCP/IP gateway and if the maker of the gateway has provided appropriate driver software, MS-DOS Kermit can connect at high speeds to any host on the TCP/IP network. For example, to get through the Interlan gateway using Kermit, run Kermit from Interlan's TELNET program by issuing the command:

`telnet` *`hostname`* `Int14h-kermit`

and then give Kermit the commands SET PORT BIOS1 and CONNECT.

If you have Novell/Excelan's TELAPI program, part of LAN WorkPlace for DOS, give the Kermit command SET PORT TELAPI *Internet-host-number* (host name lookup is not available) and then CONNECT to open an 8-bit TELNET connection; for example:

```
MS-Kermit>set port telapi 128.59.39.2
MS-Kermit>connect
```

Alt-B (or whatever key Kermit's `\Kbreak` verb is assigned to) works by sending a TELNET Interrupt Process message.

The IBM LAN Asynchronous Connection Server

EBIOS functions for terminal emulators operating on the IBM Token Ring network or Ethernet networks using the IBM NETBIOS protocols are supplied (along with MS-DOS Kermit itself) with the IBM LAN Asynchronous Connection Server (LANACS) versions 1.01 and 2.0. To use Version 1.01 of LANACS with Kermit, load `EBIOS.SYS` in your `CONFIG.SYS` file. For Version 2.0 run the `EBIOS.COM` program before operating Kermit.

The use of the LANACS REDIRECT program is optional. If you want to register a specific name for the Kermit terminal on the network, use the LANACS REDIRECT program. For

other reasons to use it, as well as more specifics on the use of the EBIOS programs, refer to the IBM LAN Asynchronous Connection Server Installation and Configuration guide for your version of LANACS.

If you decide not to use the EBIOS REDIRECT program, start Kermit and tell it to SET PORT EBIOS *n name*, where *n* is a digit 1 through 4 to simulate COM1 through COM4, and *name* is the name of the desired port or hunt group on the server. Example:

```
MS-Kermit>set port ebios 1 modem-2400
```

Then SET SPEED and PARITY to the desired values (the default parity is NONE to use 8 data bits), issue the CONNECT command, and you're on your way. If you omit the *name* in the SET PORT EBIOS command, a previous session is assumed, and will be resumed. A BREAK may be sent by pressing Alt-B (or whatever key you have assigned `\Kbreak` to). The HANGUP command also works on the server's serial port.

Whether you use the EBIOS/LANACS REDIRECT program or not, after you have issued the SET PORT command you will have to issue the SET SPEED and SET PARITY commands. Kermit relays these parameters across the network to the server. Because both workstation and server use NETBIOS to talk to the network software, this configuration can usually operate across any local area network that uses NETBIOS protocols.

AT&T StarLAN and StarGROUP

StarLAN and StarGROUP are AT&T's networking products, and MS-DOS Kermit supports this network method too. MS-DOS Kermit is the preferred method of logging in from PCs to AT&T UNIX machines running StarGROUP software.

First start AT&T StarGROUP on the PC and then start Kermit. The NETBIOS program is started automatically as part of StarGROUP. The Kermit command SET PORT NETBIOS *name* opens a connection to a remote UNIX computer that is running StarGROUP and whose NETBIOS node name is *name*. Use MS-DOS Kermit's CONNECT command to establish a terminal session, and transfer files in the normal Kermit way. If a network host is also set up as a DOS file server, its disk drives are available to Kermit for normal DOS file operations, including file transfer.

If the UNIX host provides AT&T's FACE menu system, the soft key legend displayed at the bottom of the screen is supposed to align with the PC keyboard function keys F1 through F8. The eight keys, from left to right, are expected to behave like DEC PF1 (GOLD), PF2, PF3, DECF5, DECF6, DECF7, and DECF8. The first four are the normal assignments by Kermit, and the latter four can be assigned with Kermit's SET KEY command. To use these keys with the menu system, put these commands in your MSKERMIT.INI file:

```
set key \4425 \kdecPrev     ; Enhanced keyboard gray Page Up
set key \4433 \kdecNext     ; Enhanced keyboard gray Page Down
set key \4434 \kdecInsert   ; Enhanced keyboard gray Insert
set key \4435 \kdecRemove   ; Enhanced keyboard gray Delete

def UNIX set key \270 \8,set key \320 \kdecf6,-
  set key \322 \kdecf8, set term arrow cursor
```

and use the UNIX command macro whenever you want to communicate with the UNIX system through StarLAN.

StarGROUP also allows PCs on StarLAN to use a UNIX computer's serial ports as if they were local to the PC. To do this, load the AT&T-supplied `EBIOS.SYS` program in your PC's `CONFIG.SYS` file (as a device driver—this requires you to reboot your PC after editing `CONFIG.SYS`), or run `EBIOS.COM` as a terminate-and-stay-resident program (EBIOS -U unloads the TSR). Do *not* run AT&T's REDIRECT program or the `STARTEBI.BAT` file. The functions of these programs are already built into Kermit. Start Kermit and tell it to SET PORT EBIOS *n name*, where *n* is a digit 1 through 4 to simulate serial port COM1 through COM4, and *name* identifies a hunt group on the UNIX host computer. The SET SPEED and HANGUP commands work on the server's port just as they would on a local serial port, and Alt-B (or whatever key you have assigned `\Kbreak` to) sends a real BREAK signal.

Digital Equipment Corporation DECnet

DECnet-DOS, also called DECnet-PCSA and PATHWORKS for DOS, is the PC end-node version of Digital Equipment Corporation's DECnet networking method, which can form networks of local to worldwide dimensions. Among the features of DECnet-DOS is a terminal emulator named SETHOST, named after the VAX/VMS command SET HOST, that makes a connection to any machine on the network. MS-DOS Kermit replaces that program. Once DECnet-DOS software is running on the PC and either CTERM or LAT has been loaded, start MS-DOS Kermit and give it the commands SET PORT DECNET *nodename* and CONNECT, and see a login prompt from the remote DECnet host.

DECnet-DOS provides two pathways for terminal connections: CTERM (Command Terminal) and LAT (Local Area Transport). CTERM works to any VAX near or far, while LAT is for connections on the local Ethernet. Load one program or the other depending on distance or what the local site requires. MS-DOS Kermit transparently tries LAT first (it's faster) and then, if necessary, the CTERM protocol. HANGUP and BREAK (`\Kbreak`, Alt-B) work here too.

The SET PORT DECNET command can also have an extra field for the password of a LAT server: SET PORT DECNET *node password*, for management and security purposes. SET PORT DECNET * shows a list of all the LAT servers on the network.

Other PC Networks

If you have a PC network other than Novell, chances are that MS-DOS Kermit will work in your environment too.

Ungermann-Bass Net/One

To use Kermit on an Ungermann-Bass Net/One network, activate the network and then Kermit. The command SET PORT UB-NET1 followed by CONNECT invokes the Net/One command interface to form connections to hosts. Kermit keyboard verbs `\Knethold` (normally *Alt-n*) and `\KBreak` (normally *Alt-b*) can be used to regain the attention of the Ungermann-Bass network command interface again or to send a BREAK signal across the network, respectively.

Intel OpenNET

To use MS-DOS Kermit with Intel OpenNET, run the OpenNET NETBIOS program and then start Kermit. The Kermit command SET PORT OPENNET *hostname* starts a connection to the remote host, and SET PORT OPENNET without a hostname sets MS-DOS Kermit up as an OpenNET server or resumes an earlier connection, similar to SET PORT NETBIOS.

3Com Networks

BAPI is the name for the Bridge Applications Programmer Interface to 3Com networks. It simulates a serial port over a fast network path. Start the network and then Kermit. The commands SET PORT 3COM and then CONNECT start the process, and the BAPI command menu should appear so you can select a host or modem. You can return to the BAPI menu during terminal emulation by pressing *Alt-n* (the `\Knethold` key). The 3Com BAPI interface is also supported by several other vendors of communications products.

TCP/IP Networks

TCP/IP is a widespread networking method, linking machines together locally and around the world. Kermit supports TCP/IP networks in two ways: with its own built-in TCP/IP network support in conjunction with an external packet driver, and with its BIOS interface in conjunction with external TCP/IP network software. Using Kermit on a TCP/IP network has distinct advantages over the normal TCP/IP utilities for terminal emulation (TELNET) and file transfer (FTP). You can use the same familiar software for both serial and network communication, rather than switching between different packages, and Kermit's powerful script programming and character-set translation features (found in few, if any, other TCP/IP communication packages) can be used to automate interactions between network hosts.

Kermit's Built-In TCP/IP Network Support

As of version 3.11, MS-DOS Kermit contains built-in support for TCP/IP networking. If your PC has a network adapter board connected to a TCP/IP network and an accompanying Ethernet *packet driver*, Kermit will do the rest. A packet driver is a small piece of software that takes care of the network adapter board and provides a standard interface to software application packages.[17] This allows multiple applications to use the same board, even at the same time. For example, you can log in from Kermit to a remote TCP/IP UNIX host and transfer a file between UNIX and your PC network's file server even if the network protocols are entirely different. The packet driver handles the TCP/IP traffic and your local PC network traffic simultaneously. Packet drivers are provided by the vendors of some network boards, and most packet drivers are available at no cost from network servers.[18] If you have a network board but lack a packet driver for it, contact the board vendor or, if you have access to the Internet, use FTP to get the packet driver files from host SUN.SOE.CLARKSON.EDU at Clarkson University. If you need to buy a network board, be sure to select one that comes with a packet driver. Packet drivers are available for Ethernet, Token Ring, ARCNET, and other network technologies. To use Token Ring, ARCNET, or other networks than Ethernet with Kermit, you must use an Ethernet similation packet driver.

Packet Driver Installation

Before you can use MS-DOS Kermit on a TCP/IP network, you must install your network adapter board and packet driver. Follow the directions that came with your network board. The packet driver is a program on your PC's disk. Most packet drivers are loaded as terminate-and-stay-resident (TSR) programs with command line options specific to each type of network board, such as:

```
C> C:\NETWORK\WD8003E 0X60 7 0X280 0XCA00
```

This command can be typed at the DOS prompt or added to your `AUTOEXEC.BAT` file. Other packet drivers are "device drivers" that must be loaded from your `CONFIG.SYS` file, as in this example for a Cabletron E3010 twisted-pair Ethernet board:

```
DEVICE=C:\CBLTRON\CSIPD_E.EXE
```

This automatically loads the packet driver whenever you start your PC.

MS-DOS Kermit is automatically configured to use the packet driver. You should configure all your other network software (for example, the Novell network shell) to use it too. Consult your network manager for details.

---

[17]The packet driver specification is available from FTP Software, Inc. and has been widely adopted by network board manufacturers and PC software developers.

[18]The main collection of copyrighted but free packet drivers is maintained by Clarkson University in Potsdam, New York.

TCP/IP Setup

Before you can establish a TCP/IP network connection, you must use SET TCP/IP commands to tell Kermit about your TCP/IP networking environment. Here are the commands you will need. See your network manager to find out the values you must use:

SET TCP/IP ADDRESS *IP-address*

IP is the network layer of the TCP/IP protocol. Different computers on a TCP/IP network send messages to each other using IP addresses. An IP address is a series of four decimal numbers, each in the range 0 to 255, separated by periods and with no leading zeros, for example:

```
128.59.39.2
```

Your network manager must assign a unique IP address to your PC; don't make one up yourself! Kermit must know your PC's IP address before it can communicate with other computers on the network, so this command is *required*.

SET TCP/IP SUBNETMASK *IP-address*

This command determines the scope of addressing for your immediate local area network. The default value is 255.255.255.0. This means that the final 8 bits of an IP address are specific to your physical network, allowing approximately 250 nodes. The value 255.255.254.0 would be used for 9-bit addressing, allowing about 500 nodes, and so forth. Check with your network manager to find out the correct subnet mask for your network.

SET TCP/IP PRIMARY-NAMESERVER *IP-address*

There are thousands of computers on the worldwide IP network (the Internet). No single computer can be expected to keep track of all their names. Instead, *name servers* are established at central sites, and other computers (such as your PC) send special messages asking them to translate host names into IP addresses. Use this command to tell Kermit the IP address of the name server in your local area network. If you do not specify a nameserver address, you cannot use IP hostnames in Kermit commands, but you can still use numeric IP host addresses.

SET TCP/IP SECONDARY-NAMESERVER *IP-address*

Use this command to tell Kermit the IP address of the nameserver to use if the primary nameserver is unreachable.

SET TCP/IP GATEWAY *IP-address*

Your local area network is connected to your organization's backbone network and to the worldwide Internet by a *gateway*. If you need to communicate beyond your local area network, you must specify a gateway address.

SET TCP/IP DOMAIN *domain-name*

This is the IP domain name for your organization, for example "columbia.edu" for Columbia University, "dec.com" for Digital Equipment Corporation, and "ibm.com" for International Business Machines Corporation.

The SET TCP/IP commands would normally go in your `MSKERMIT.INI` file.

Establishing, Using, and Releasing the TCP/IP Connection

After you have loaded your packet driver and set up your TCP/IP network environment, you can start your network session with this command:

SET PORT TCP/IP *ip-host-name-or-address*

You can use a host name, such as `nic.ddn.mil`, if you have specified a name server address. If you don't have a name server or if your name server does not respond, you can use an IP address. If the connection can't be made within a reasonable amount of time, Kermit gives you a message like "host unreachable." Examples:

```
MS-Kermit>set port tcp/ip nic.ddn.mil
MS-Kermit>set port tcp 192.67.67.20
```

Once the connection is established, Kermit uses the TCP/IP TELNET protocol to communicate with the remote host. TELNET is a "virtual terminal" protocol that allows two computers with a network connection to behave like a terminal connected to a computer. The TELNET protocol is operational during all Kermit commands that communicate with the remote host. Such items as echoing and terminal type are negotiated automatically according to MS-DOS Kermit's current LOCAL-ECHO and TERMINAL TYPE settings.

Kermit's use of the TELNET protocol should be transparent to you, so a TCP/IP connection behaves just like a serial port (but usually faster). One small point is worth mentioning, however: the character value 255 (an 8-bit byte with all 8 bits set to 1) is the TELNET protocol's "escape character," used internally by Kermit for exchanging messages with the remote host's TELNET server. If you ever need to send this character as data, for example as an international character, Kermit automatically doubles it, which makes the remote TELNET server accept one copy of it as data.

All of Kermit's commands remain effective during a TCP/IP connection except the ones that are concerned with serial port operation, such as SET SPEED, SET FLOW RTS/CTS, SHOW MODEM, etc. If you need to send a BREAK signal to the remote computer during terminal emulation, just press Alt-B or whatever key you have assigned the `\Kbreak` Kermit verb to; this sends a TELNET protocol BREAK.

The details of the TELNET connection vary from host to host. In many cases, you can have successful terminal emulation, script execution, and file transfer with a parity setting of NONE. If these operations fail, however, you should set your parity to SPACE. The maximum packet length and window size usable during file transfer also varies according to the connection and the characteristics of the remote host. Experiment to find the most efficient combination.

The normal way to close a TCP/IP session is to log out from the remote host. This usually breaks the connection automatically. You can also close the connection by giving the HANGUP command, and the connection is closed automatically when you EXIT from Kermit.

External TCP/IP Network Software

Kermit can also be used as the terminal emulator in TELNET connections for certain external TCP/IP products. The TCP/IP product must provide a program to be used in place of TELNET, which handles the network communications and the TELNET protocol, but which makes itself available to Kermit through BIOS Interrupt 14, so Kermit can still control the keyboard and the screen. This lets MS-DOS Kermit replace the normal TELNET program, and also transfer files over the same connection. To use a BIOS Interrupt 14 interceptor, Kermit must be given this command:

```
MS-Kermit>set port bios1
```

Commercial products that support the Interrupt 14 type of connection include: FTP Software Inc.'s TNGLASS routine running with their kernel software, Interlan's TCP/IP Gateway for Novell Networks (described above), Wollongong's WIN/TCP and WIN/ROUTE for DOS, and Novell's Excelan LAN WorkPlace for DOS. The last one provides INT 14h service as well as 3Com BAPI, TELAPI, and other interfaces usable by MS-DOS Kermit. A small interface program, such as TNGLASS, is given the name of the remote host and starts Kermit when the host responds. Here, for example, is a DOS batch program you can use to invoke Kermit "under" TNGLASS:

```
@echo off
tnglass %1 -e kermit set port bios1, stay
```

Call this file (for example) TNKER.BAT, store it anywhere in your DOS PATH, and run it by typing "tnker" followed by the IP host name or number. Then use ordinary Kermit commands like CONNECT to begin a terminal session with the host. EXIT from Kermit ends the session; Kermit's PUSH or RUN commands keep it alive.

Similarly, for Wollongong WIN/TCP, use a batch file like this:

```
@echo off
telnet -e kermit %1
```

where %1 is the host name or address, or like this:

```
@echo off
rlogin %1 -e -l %2
kermit
```

where %2 is your remote username. When Kermit starts, issue the SET PORT BIOS1 command, or put this command in your MSKERMIT.INI file.

MS-DOS Kermit supports a variety of Interrupt 14h standards, such as native BIOS, 3Com BAPI, IBM EBIOS/LANACS, Novell TELAPI, and InterConnections/Novell TES, plus examples of the Ungermann Bass Interrupt 6Bh standard such as UB-Net/One and Novell NASI/NACS.

Network Connection Status when Exiting Kermit

MS-DOS Kermit terminates most network connections when you EXIT or QUIT from Kermit to DOS. The exception is SET PORT BIOS connections (except for TCP/IP), which stay active even after Kermit exits because the network TSR is still loaded in memory. Some of these connections can be placed on hold by talking to the network terminal control program and making the request; these networks include EBIOS/StarGROUP, EBIOS/LANACS, TES, and Novell NASI/NACS. When Kermit is started again later the same terminal control program can be requested to resume an old connection. The details of the terminal control conversation vary from network to network, so check the manual for your network.

Chapter 17

MS-DOS Kermit Command Reference

This chapter summarizes MS-DOS Kermit 3.11 commands and features, including those that were not discussed in the text.

MS-DOS Kermit can be run interactively, from a batch file, as an "external" DOS command, or with redirected or piped standard input and output, and it can run in various PC windowing environments, such as Microsoft Windows.

When MS-DOS Kermit starts, it automatically executes any commands found in the file `MSKERMIT.INI` in the current disk and directory or DOS PATH, or in the file specified by `-f` *filename* on the Kermit command line.

Bugs and Patches

The binary, executable Kermit program included on the diskette that came with this book can have corrections, or *patches*, applied to it in case bugs are discovered in the program after it was released. These patches can be applied only to the binary program as distributed, not to any other version. Patches are distributed in the form of a short text file called `MSKERMIT.PCH`. This file should be stored in the same disk and directory as your `MSKERMIT.INI` file. To apply the patches, add a line that says "patch" to your `MSKERMIT.INI` file. If there is anything wrong with the patch file or if it does not agree in any way with your Kermit version, the patches are not applied. Obtain legitimate patch files only from Columbia University.

Interactive Operation

To run MS-DOS interactively, you invoke the program from DOS command level by typing its name, usually `kermit`. When you see the command's prompt, `MS-Kermit>`, you may type Kermit commands repeatedly until you are ready to exit the program. The command EXIT or QUIT returns you to DOS.

While typing commands, use the Backspace key to erase the character most recently typed, *`Ctrl-W`* to delete the most recent word, or *`Ctrl-U`* to delete the entire command, and start the command by pressing the Enter key (carriage return) or *`Ctrl-L`*. A question mark typed at any point in a command (except in a filename or character string) will give you a brief hint about what's expected at that point. Pressing the Esc key will complete the current word, if possible, and position the cursor at the next command field. If completion is not possible, Kermit will beep. You can cancel any command during its execution by typing *`Ctrl-C`*.

Piped Operation

MS-DOS Kermit's command-level input and output may be redirected or piped using the normal DOS mechanisms:

```
C>kermit < commands.tak
C>kermit > commands.log
C>kermit < commands.tak > commands.log
C>Kermit < commands.tak | sort > commands.srt
```

Terminal emulation is not affected by redirection or piping.

Command Line Invocation

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

```
C>kermit set port 1, set speed 9600, connect
```

Several commands may be given on the command line, separated by commas. Help and completion are not available when Kermit commands are given on the command line. MS-DOS Kermit will exit back to DOS after completing the specified commands unless you include the STAY command on the command line:

```
C>kermit connect, stay
```

If you want Kermit to execute a different initialization file upon startup, you can create a new file with a unique name and specify its name on the command line:

```
C>kermit -f a:\test\tuesday.ini
```

If you want to run Kermit *au naturel* with all its "factory settings," you can specify the null (empty) initialization file:

```
C>kermit -f nul
```

If `-f` is the only command line option, STAY is implied, and the `MS-Kermit>` prompt will appear.

Remote Operation

MS-DOS Kermit can be put into server mode and accessed from other computers. Furthermore, the MS-DOS CTTY command allows an MS-DOS system to be used from a terminal connected to its communication port, for example CTTY COM1. You can then use DOS and Kermit from a terminal or computer connected to the PC's COM1 device.

Running MS-DOS Kermit in Windowing Environments

MS-DOS Kermit does not have pop-up or pull-down menus, and it does not support command entry by mouse clicks. But it is written in a "windows-aware" manner to let it run under OS/2, Microsoft Windows, or Quarterdeck DesqView. This section contains hints about how to configure these environments for MS-DOS Kermit. These are only suggestions. The fine-tuning is up to you.

IBM OS/2

MS-DOS Kermit works under OS/2, but only in DOS compatibility (full screen) mode. Simply issue the OS/2 command:

```
SETCOM40 COM1=ON
```

before starting Kermit to enable the use of the COM1 communication port.

Microsoft Windows 2.0

MS-DOS Kermit works well in a Microsoft Windows 2.0 window: it uses Windows fonts, it works with mice, accepts cut-and-paste material, and shrinks to an icon (the word KER in a small box). Kermit can transfer a file in one window while you do other work in other windows. You can have multiple copies of Kermit going simultaneously in different windows, as long as they don't try to use the same communication port. To use MS-DOS Kermit as a Windows 2.0 application, use the Windows PIFEDIT program to create `KERMIT.PIF`, a Program Information File for Kermit as follows:

Program Name: `KERMIT.EXE` or whatever you have named the Kermit program file. *Program Title:* MS-DOS Kermit 3.11. *Program Parameters:* Command line arguments for Kermit, normally blank. *Initial Directory:* The directory, if any, to CD to when starting Kermit. *Memory Requirements:* 220 KB Required, 400 KB Desired. *Directly*

Modifies: Uncheck all boxes. *Program Switch:* Check the Text box. *Screen Exchange:* Check the Graphics/Text box. *Close Window on Exit:* Check the box.

Although Kermit does write directly to screen memory it does so in a "windows aware" manner. You can check the COM1 or COM2 boxes or them leave empty even though Kermit accesses the serial port hardware directly. Kermit will send an Xoff if left running as an icon, but it will run smoothly while sharing the screen with other tasks. Terminal emulation efficiency is limited by Windows' character drawing speed. Graphics are done as if you had a monochrome adapter. Screen dumps are of Kermit's underlying full screen.

If you check the "modifies memory" box or some of the other boxes (or if you don't have a `KERMIT.PIF` file at all), Kermit takes over the whole screen, Windows becomes inactive until Kermit exits, and Windows features will no longer work in Kermit. Kermit will run much faster, however, and Tektronix graphics will work normally.

If Windows complains that it does not have enough memory to run Kermit, you can reduce Kermit's memory requirements by allocating less memory for rollback screens. Kermit's default number of rollback screens is 10, and each rollback screen takes about 4K of memory (more or less depending on the type of display adapter you have). With the default 10 rollback screens, Kermit needs about 220K of memory. You can reduce this to a minimum of about 180K by getting rid of some or all of your rollback screens. Put a line like SET KERMIT=ROLLBACK 0 in your `AUTOEXEC.BAT` file, or type this command to DOS before starting Windows.

If you have trouble accessing a serial communication port from within Windows, make sure you have an appropriate entry for the port in the `[ports]` section of your `WIN.INI` file, for example:

```
COM1:=2400,n,8,1
```

where COM1 is the name of the device, 2400 is the speed in bits per second, "n" means no parity, 8 means eight data bits, and 1 means one stop bit.

Microsoft Windows 3.0

Microsoft Windows 3.0 allows Kermit to run as described for Windows 2.0, but only if Windows 3.0 is running in "enhanced mode," which requires an 80386-class or higher computer. If you have a lower-class computer (80286 or lower, such as a PC/AT or a PS/2 Model 30 or 50), or you are running Windows 3.0 in Standard or Real mode, non-Windows DOS applications such as Kermit cannot be run in a window. This is a limitation of Windows 3.0. In this case, do not create a `KERMIT.PIF` file, and Windows will run Kermit in full-screen DOS mode if it has enough memory and swap space.

If you have an 80386 or above running Windows 3.0 in enhanced mode, you can set MS-DOS Kermit up similarly to the method described for Windows 2.0. Follow the directions for Windows 2.0 and then return here for the additional options:

Display Usage: Check "Windowed" if you plan to be using only text terminal emulation. If you need to do Tektronix graphics terminal emulation, check "Full Screen" instead. *Execution:* Check Background and uncheck Exclusive. *Close Window on Exit:* Check.

Now click on the "Advanced" button to set the following enhanced mode options:

Multitasking: Check "Detect Idle Time." Set Background Priority to 50 and Foreground priority to 100. *Memory Options:* 0 KB EMS and XMS memory required, 1024 KB limit, not locked; MS-DOS Kermit does not not use high memory. *Display Options:* High Graphics Video Memory and Monitor Ports. Check "Emulate Text Mode" and "Retain Video Memory". Kermit uses all of graphics memory to save graphics images. *Other Options:* Check *Allow Fast Paste*, do not check *Allow Close when Active. Reserve Shortcut Keys:* None (uncheck all boxes). *Application Shortcut Key:* None.

True graphics can be done within a window under Windows 3.0; you can even scroll the window around to see various parts of Kermit's graphics screen. But when you (or the host) attempt to switch Kermit back to text mode, Windows becomes concerned about where to put Kermit's graphics memory and advises you to run Kermit in full-screen mode, and does not save the graphics screen. This problem does not occur if Kermit does not enter graphics mode or if it is run in full-screen mode under Windows.

Quarterdeck DesqView

When configured properly for DesqView, Kermit works approximately as described for Windows 3.0 enhanced mode: concurrent operation with other PC applications, two Kermits running simultaneously on different communication ports, and full graphics capability. Here is a sample configuration:

Program Name: MS-DOS Kermit v3.11. *Keys to Use on Open Menu:* KE. *Memory Size (in K):* 220. *Program:* `KERMIT.EXE`. *Parameters:* Command line arguments for Kermit. *Directory:* Disk and directory to CD to upon startup, if any. *Writes text directly to screen:* Say "No" (not true, but MS-DOS Kermit is DESQview-aware and covers up nicely). *Displays graphics information:* No (not true, but causes DESQview to put Kermit into full-screen mode while graphics are executed and then return to windowed mode after switching back to text terminal emulation). *Virtualize text/graphics (Y, N, T):* T or Y. *Uses serial ports (Y, N, 1, 2):* Y. *Requires floppy diskette:* N. *Text Pages:* 1. *Graphics pages:* 0. *Initial Mode:* 3. *Interrupts:* 00 to FF. *Window Position:* Maximum Height: 25, Starting Height: 20, Starting Row: 5, Maximum Width: 80, Starting Width: 40, Starting Column: 5. Or choose your own window sizes. 132-column mode causes DESQview to

switch to full screen. *Allow Close Window Command:* Y. *Uses its Own Colors:* N. *Runs in Background:* Y. *Uses Math Coprocessor:* N. *Keyboard Conflict:* 0. *Share CPU when Foreground:* Y. *Share EGA when Foreground/Zoomed:* Y. *Can be Swapped Out:* Leave blank. *Protection Level:* 3.

DOS Emulators

MS-DOS Kermit is known or reported to run under DOS emulators such as SoftPC from Insignia Solutions, Inc. on NeXT and other workstations. On the NeXT, SoftPC must be configured to use the NeXT's modem ports: Click on Info in the SoftPC menu, click on Preferences in the Info menu, click on Ports in the Preferences menu, and then tell SoftPC that your COM1 port is Serial A, using the device name `/dev/cua`. If you need to use communication software (such as C-Kermit) outside of SoftPC, you must reconfigure your serial port in SoftPC to be "none." When properly configured, Kermit runs in a NeXT window. Screens can be captured and printed using the Print menu, and screen material can be cut and pasted using the mouse and the Edit menu.

Batch Operation

Like other DOS programs, MS-DOS Kermit may be run from within batch programs by including Kermit command line arguments. If you invoke it without command line arguments, it runs interactively, reading commands from the keyboard and not the batch file. When Kermit exits, batch processing continues to the end of the batch file. An ERRORLEVEL number is returned by Kermit to assist batch file controls, as shown in Table 17-1. Here is a sample DOS batch program, `SEND.BAT`, that runs Kermit to send a file and then reports success or failure to the user:

```
@echo off
if exist %1 goto send
echo File %1 not found
goto end
:send
echo Sending %1...
kermit -f nul, set display quiet, send %1
if errorlevel 1 goto bad
echo File %1 sent successfully.
goto end
:bad
echo File %1 could not be sent.
:end
```

This batch program can be used to send any file by including the filename on the DOS command line:

```
C>send oofa.txt
```

Table 17-1 MS-DOS Kermit Return Codes

ERRORLEVEL	*Kermit Session Status*
0	Entirely successful operation
1	A SEND command completed unsuccessfully
2	A RECEIVE or GET command completed unsuccessfully
4	A REMOTE command completed unsuccessfully
3, 5, 6, 7	Combinations (addition) of the above conditions
8	TAKE file error
16	Many reasons
256	User intervention (Ctrl-C)

To set a Kermit variable from within the batch program, double the percent sign, as in this example:

```
kermit define \%%f %1, take auto.scr
```

See the batch section of your DOS manual for details about DOS batch programming.

Nonstandard Communication Ports

If your PC has an 8250 or equivalent (16450, 16550A) UART (the standard serial port hardware for IBM PCs and PS/2s), MS-DOS Kermit controls it directly for high-speed interrupt driven communication. In order to do this, Kermit must know the address and interrupt request line (IRQ) number of the port. Serial ports and internal modems should work correctly with Kermit if they are installed as COM1 or COM2. The industry standard puts COM1 at address 03F8 (hex) on IRQ 4, and COM2 at 02f8 on IRQ 3. But there has been very little agreement about a standard for COM3 or COM4 until the IBM PS/2. On the PS/2, COM3 is at 3220 and COM4 is at 3228, and both are on IRQ 3. Third-party boards or "semi-clone" PCs often use different addresses and sometimes different IRQ numbers.

MS-DOS Kermit's SET PORT command tries its best to locate your port hardware and identify its IRQs automatically. But when your PC or communication hardware fails to follow the standards and conventions known to Kermit, you won't be able to communicate with the selected device. To overcome this obstacle, MS-DOS Kermit lets you tell it the address and IRQ number of a serial port. You will never have to use this command unless you have a very nonstandard configuration. *Use this feature with extreme caution!* The command is SET COM*n*, described on page 228.

DOS Environment Variables for Kermit

In your AUTOEXEC.BAT file, you may include a line like:

```
SET KERMIT=command;command;...
```

The commands are special Kermit startup configuration parameters. These include:

COM1–**COM4**
Address and IRQ number of nonstandard COM ports, as in the SET COM*n* command.

INPUT-BUFFER *length*
The number of characters for INPUT and REINPUT commands to retain. The number can be as high as 65535 if your PC has enough memory.

ROLLBACK *length*
The number of screens to retain in the rollback buffer; 10 is the default, and 0 eliminates rollback. The number can be as high as 130 if your PC has enough memory.

Here is a sample AUTOEXEC.BAT entry:

```
set kermit=rollback 20;input-buffer 256;com3 \x03e8 \2
```

You can examine your KERMIT environment like this:

```
MS-Kermit>echo \$(kermit)
```

You can also access any other DOS environment variable using \$(*name*), where *name* is the name of the environment variable, for example \$(path) or \$(comspec).

Kermit runs the program denoted by the DOS environment variable SHELL (if found, otherwise by COMSPEC) when executing PUSH, RUN, TYPE, and similar commands. If SHELL and COMSPEC are not defined, Kermit uses COMMAND.COM from the boot device.

File Specifications

In all commands, file specifications may include fully qualified DOS paths, including device specifications. Allowable wildcard characters are * (match from here to end of field) and ? (match a single character), as in DOS. The only difference is that # must be used instead of ? to match the first character of a filename, in order to allow ? to give help in that position. Here are some examples:

```
OOFA.TXT          (File in current disk and directory)
\TOM\OOFA.TXT     (File in directory \TOM\ on current disk)
A:OOFA.TXT        (File in A disk's current directory)
A:\TOM\OOFA.TXT   (File in directory \TOM\ on A disk)
*.TXT             (All files of type .TXT)
OOFA.*            (All files with name OOFA)
#.*               (All files with a one-character name)
*.?               (All files with a one-character type)
```

Interrupting a File Transfer in Progress

The following keys may be used during file transfer or REMOTE commands:

X *or* **Ctrl-X**
Stops transferring the current file and goes on to the next one, if any.

Z *or* **Ctrl-Z**
Stops transferring the current file and doesn't send any further files.

E *or* **Ctrl-E**
Sends an error packet to the remote Kermit program to make it quit the current protocol operation.

C *or* **Ctrl-C**
Returns to MS-DOS command level immediately.

Q *or* **Ctrl-Q**
Send a Ctrl-Q (Xoff) character.

Enter
Tries to break a protocol deadlock by retransmitting the most recent packet. Press the Enter key when nothing seems to be happening.

Notation in Command Descriptions

The following notation is used in this command summary:

parameter
Replace this with an actual number, filename, or whatever type of item is called for.

number
Replace with an actual decimal number. In many cases, octal or hexadecimal numbers may also be specified, prefixed by \o or \x.

filespec
An MS-DOS file specification, which may include disk and directory.

hh:mm:ss
Time of day in 24-hour notation, less than 12 hours from current time.

[*parameter*]
An optional parameter, which may be omitted. (You don't type the brackets.)

{A,B,C}
Choose one of the items listed. Type only one of the listed items. (You don't type the braces or commas.)

[{A,B,C}]
Optionally choose one of the items listed.

`text`
In examples, text that you type is underlined. When you finish typing the text, press the Enter key.

`Ctrl-X`
In examples, this means hold down the Ctrl key and press the X key (X can also be any other key). Do not push Enter afterward.

`Alt-X`
In examples, this means hold down the Alt key and press the X key (X can also be any other key). Do not push Enter afterward.

`Ctrl-]X`
In examples, this means hold down Ctrl and press the right bracket (]) key, then let go of both these keys and press the X key (X can also be any other key).

Comments and Continuation

In TAKE command files only, commands may have trailing comments preceded by a semicolon, for example:

```
set cursor block  ; I like a big cursor
set term bell vis ; and a visible bell
```

All characters starting with the first semicolon in each line of a TAKE file are ignored by Kermit. To include an actual semicolon in a command within a TAKE file, precede it with a backslash, for example:

```
get oofa.exe\;2
```

Kermit commands can be continued onto the next line by ending the continued line with a hyphen, for example:

```
echo this is not really -
such a long line.
```

Continuation is allowed in command files and macro definitions (but it is not required; commands can be up to 256 characters long). If you actually need to end a line with a hyphen, use backslash notation (`\45`). If you use continuation at the end of a trailing comment, the following line is treated as part of the comment.

Backslash Notation

Backslash notation can be used to specify a number or a character in any command where a single character must be specified, such as `SET ESCAPE \29`. Backslash is also used to introduce variable names and other special quantities in Kermit commands. Any letters following the backslash may be either uppercase or lowercase. Table 17-2 shows MS-DOS Kermit's backslash notation. Whenever there is ambiguity, you can resolve it by using curly braces. The left brace goes immediately after the backslash, and the right brace terminates the backslash code:

```
\{123}                  ASCII character 123 (left brace itself)
\{12}3                  ASCII formfeed (12) followed by "3"
\{Kexit}                Kermit keyboard verb \Kexit
```

See Table I-4 for a list of Kermit verbs.

Table 17-2 MS-DOS Kermit Backslash Notation

Code	*Meaning*
`\123`	(up to 3 decimal digits)—A decimal number
`\d123`	(up to 3 decimal digits)—A decimal number
`\o123`	(up to 3 octal digits)—An octal (base 8) number
`\x12`	(up to 2 hexadecimal digits)—A hexadecimal (base 16) number
`\92`	A literal backslash
`\{}`	For grouping, e.g., `\{27}3` = ESC 3
`\;`	Include a semicolon in a TAKE-file command or macro definition
`\%`	Introduce a Kermit variable, `\%1`, `\%2`, ..., `\%a`, `\%b`, ... `\%z`
`\K`	A Kermit keyboard verb, like `\Kexit` or `\Kbreak`
`{\K}`	A Kermit macro name, to be used as a keyboard verb
`\b`	Send a BREAK (OUTPUT command only)
`\255`	Shorthand for CRLF or LFCR (INPUT command only)
`\CD`	Carrier Detect RS-232 signal (WAIT command only)
`\DSR`	Data Set Ready RS-232 signal (WAIT command only)
`\CTS`	Clear To Send RS-232 signal (WAIT command only)
`\M(name)`	Refer to a macro's definition, e.g. `\m(number)`
`\V(name)`	Introduce a built-in variable like `\v(date)`
`\$(name)`	Introduce a DOS environment variable like `\$(comspec)`

MS-DOS Kermit Commands

This section lists all of MS-DOS Kermit's commands alphabetically. Subsequent sections give details about the IF, REMOTE, SET, and SHOW commands.

-F *filespec* (from DOS command line only)
Uses *filespec* instead of `MSKERMIT.INI` as the initialization file. `kermit -f nul` runs Kermit without any initialization file at all. The `-f` command can be issued only on the Kermit command line. Example:

```
C>kermit -f midnight.ini
```

ASK *variable-name prompt string*
Prints the prompt on the screen and sets the value of the variable to be what the user types in response. Example:

```
MS-Kermit>ask \%p Password:
Password: secret
MS-Kermit>
```

The value of `\%p` is now "secret" as defined by the user.

ASKQ *variable-name prompt string*
"Ask Quietly"—like ASK, but what the user types does not echo. Example:

```
MS-Kermit>ask \%p Password:
Password: ______
MS-Kermit>
```

ASSIGN *name* [*string*]
Evaluates the string by expanding all variable references it may contain, and copies the result into the definition of the macro or variable whose name is *name*. If a macro is being defined, commands within the definition should be separated by commas. Examples:

```
MS-Kermit>assign \%a Hello There!
MS-Kermit>assign friendly echo \%a \%a \%a
MS-Kermit>assign \%m I said, "\%a"
```

Here, the definition `\%m` is `I said, "Hello There!"` and not `I said, "\%a"`. If the definition string begins with `{` *and* ends with `}`, the braces are removed. If the definition string is empty, the variable or macro becomes undefined.

BYE
Shuts down a remote Kermit server, logs out its job, and exits from MS-DOS Kermit:

```
MS-Kermit>bye
C>
```

C
Special abbreviation for CONNECT:

```
MS-Kermit>c
```

CD [*path*] (or **CWD** [*path*])
Changes working directory on your PC. If *path* is omitted, this command tells you your current working directory. Unlike the DOS CD command, the Kermit CD command lets you include a disk drive letter and/or a directory name. Examples:

```
MS-Kermit>cd mupeen      (Relative directory)
MS-Kermit>cd \mupeen     (Absolute directory)
MS-Kermit>cd b:          (Disk)
MS-Kermit>cd b:\mupeen   (Disk and directory)
MS-Kermit>cd             (Show current)
```

CLEAR
Clears the communications device and INPUT command buffers. Example:

```
MS-Kermit>cle
```

CLOSE { ALL, PACKETS, READ-FILE, SESSION, TRANSACTION, WRITE-FILE }
Closes the specified file or log. All files are closed automatically when Kermit exits. Examples:

```
MS-Kermit>close p
MS-Kermit>clo all
```

CLS
Clears the PC's screen, just like the DOS CLS command.

COMMENT *text*
Does nothing. For adding comments to a TAKE command file. Example:

```
COMMENT - And now set some parameters...
```

CONNECT
Makes a terminal connection to another computer. Type the escape character followed by C to return to the `MS-Kermit>` prompt. The escape character is normally *`Ctrl-]`* (see Table I-3). Use SET ESCAPE to change the escape character, or use SET KEY to assign `\Kexit` or other Kermit verbs to the keys of your choice. `\Kexit` is also assigned to *`Alt-X`* on IBM keyboards. CONNECT may be abbreviated simply as C.

DEFINE *name* [*string*]
Copies the string (if any) literally as the definition of the named macro or variable. Variables in the string are *not* evaluated. If the string begins with "`{`" *and* ends with "`}`", the braces are removed. If no string is given, DEFINE undefines the named macro or variable. If a macro is being defined, commands within the definition should be separated by commas. Examples:

```
MS-Kermit>define unix set spe 9600, set par none, c
MS-Kermit>define \%n 1-800-555-1234
MS-Kermit>define ibm
```

If the definition includes a variable name, the variable name is included literally so it can be evaluated later, when the macro is executed:

```
MS-Kermit>define rename run ren \%1 \%2
```

Commas in the macro's definition are replaced by carriage returns unless they are surrounded by curly braces; this allows a macro to define other macros:

```
MS-Kermit>define xx define yy echo foo, echo bar, do yy
MS-Kermit>xx
bar
foo
MS-Kermit>define xx define yy { echo foo, echo bar }, do yy
MS-Kermit>xx
foo
bar
```

DELETE *filespec*
Deletes local PC files. Example:

```
MS-Kermit>delete junk.*
```

DIRECTORY [*filespec*]
Lists names, sizes, and dates of local PC files. Runs DOS's DIR command. Examples:

```
MS-Kermit>directory
MS-Kermit>directory oofa.*
MS-Kermit>dir a:
MS-Kermit>dir \chris\
MS-Kermit>dir a:\chris
```

DISABLE *function*
Restricts the specified function as indicated when in server mode: CD (disallow), DEL (current directory only), DIR (current directory), FIN (disallow FINISH, BYE, and LOGOUT), GET (current directory), HOST (disallow), KERMIT (REMOTE KERMIT commands not allowed), LOGIN (not required), PRINT (disallowed), SEND (current directory), SPACE (disallowed), and TYPE (current directory). ALL disables all these at once. Example:

```
MS-Kermit>disable all
```

[**DO**] *macro-name arguments*
Assigns the argument words (if any) to the macro arguments `\%1`, `\%2`, and so on, up to `\%9`. Assigns the name of the macro to `\%0`. Then performs the commands in a macro, substituting references to argument names (if any) by their assigned values (if any). The word DO can be omitted; you can just type the name of the macro. Examples:

```
MS-Kermit>do ibm
MS-Kermit>do dial 7654321
MS-Kermit>ibm
MS-Kermit>dial 7654321
```

ECHO *text*
Displays the *text* on the screen. Any backslash-codes, including temporary, permanent, and built-in variables, within the text are evaluated. A carriage return and linefeed is supplied automatically at the end of the text. Example:

```
MS-Kermit>echo \13\10\10\Bill, it's \v(time).  Wake up!!
```

ECHO by itself prints a blank line. Backslash codes for 8-bit values can be used to display characters from the "top half" of the current PC code page, such as international characters or line and box drawing characters.

ENABLE *function*
Allows the specified function as indicated when in server mode: CD (to any directory), DEL (any file anywhere), DIR (of any directory), FIN (FINISH, BYE, and LOGOUT), GET (file from any directory), HOST (commands allowed), LOGIN (required), PRINT (allowed), SEND (to any directory), SPACE (allowed), and TYPE (files in any directory). ALL enables all these at once. Example:

```
MS-Kermit>ena login
```

END [*n*]
When executing a command file or macro, returns immediately to the invoking level (macro, TAKE file, or `MS-Kermit>` prompt). An optional number can be included after the END command to indicate success (0) or failure (nonzero) of the TAKE file or macro; this allows IF commands to test whether the command file or macro as a whole succeeded or failed. END does nothing at command level. POP is a synonym for END. Examples:

```
end
if failure end
if failure end 1
if success end 0
```

EXIT
Exits from MS-DOS Kermit. Closes all open logs and other files. Serial port connections are left active (Kermit does not hang up). Most types of network connections are closed. If a macro named ON_EXIT is defined, Kermit executes it. QUIT is a synonym for EXIT. Example:

```
MS-Kermit>ex
C>
```

FINISH
Shuts down a remote Kermit server but does not exit from MS-DOS Kermit and does not terminate the remote host session (compare with BYE). Example:

```
MS-Kermit>fin
MS-Kermit>
```

GET *remote-filespec*
Asks a Kermit server to send the specified file(s). Use this command instead of RECEIVE when communicating with a Kermit server. To specify an alternate name to store the file under when it arrives, type `get` followed immediately by the Enter key to be prompted for remote and local filespecs separately. Examples:

```
MS-Kermit>get oofa.txt
MS-Kermit>get *.c
MS-Kermit>get
 Remote Source File: filename with spaces
 Local Destination File: spaces.no
```

To give a multiline GET command as a Kermit command-line argument, separate each line with a comma:

```
C>kermit get, filename with spaces, spaces.no
```

GOTO *label*
Transfers control to the specified label in a macro or TAKE file. The label must appear at the beginning of the line and it must start with a colon (`:`). If the label is not found in the current macro or command file, Kermit goes one level up to the macro or command file that invoked the current one and looks there, and so on, back up to top level. Command file example:

```
goto xxx
echo You won't see this!
:xxx
echo You will see this.
```

Macro example:

```
define gotest goto xxx,echo You won't see this!,:xxx,-
echo You will see this.
```

HANGUP
On serial port connections, briefly turns off the DTR modem signal (see Glossary) in an effort to break a modem or similar connection. On network connections, closes the connection. Example:

```
MS-Kermit>hang
```

HELP
Displays a brief help message about MS-DOS Kermit.

I
Special abbreviation for INPUT. Example:

```
MS-Kermit>i 10 login:
```

IF [NOT] *condition command*
In a script, executes the command if the condition is true or, if NOT is included, executes the command if the condition is not true (listed on page 222).

INPUT [*timeout*] *text*
Tries to receive the specified text from the remote system through the current communication device (port) within *timeout* seconds. Sets SUCCESS or FAILURE for subsequent IF commands. *text* may contain backslash codes. Examples:

```
input 5 Password:\32
in 7 \27[\{63}0c
```

Keyboard activity before the timeout expires causes immediate failure. The terminal emulator is *not* active during the INPUT command. See also SET INPUT.

LOG PACKETS [*filespec*]
Records file transfer packets in the specified file or device. Default file is `PACKET.LOG`. Example:

```
MS-Kermit>log packets julie.log
```

LOG SESSION *filespec*
Records your terminal session in the specified file or device. Default file is `SESSION.LOG`. Example:

```
MS-Kermit>log sess monday.log
```

LOG TRANSACTIONS *filespec*
Reports the progress of file transfers in the specified file or device. Default file is `TRANSACT.LOG`. Example:

```
MS-Kermit>log t overnite.log
```

LOGOUT
Shuts down the remote server, logs out the remote host session, but doesn't exit from MS-DOS Kermit (compare with BYE, FINISH). Example:

```
MS-Kermit>logo
```

MAIL *filespec address*
Sends the file as mail to the specified address. The remote Kermit must be in RECEIVE or SERVER mode, and must support the MAIL command. The fields of this command can be typed on separate lines. Examples:

```
MS-Kermit>mail message.txt michael@cuvma.bitnet
MS-Kermit>mail
 Local Source File: message.txt
 To: michael@cuvma.bitnet
```

OPEN { **APPEND, READ, WRITE** } *filename*
Opens the specified file in the specified mode. READ means to open an existing file for reading by the READ command; if the file does not exist, the OPEN READ command fails. WRITE means to create a new file for use by the WRITE command; if a file of the given

name already exists, it is destroyed. APPEND means to open an existing file so that WRITE commands will add new material to the end; if the specified file does not exist, a new file is created. Examples:

```
open read dialing.dir
open write a:new.txt
open append c:\logs\friday.log
```

OUTPUT *text*
Sends the specified text to the remote host as if you had typed it; *text* may contain back-slash codes. The special code `\b` sends a BREAK signal. Example:

```
output run kermit\13
```

PATCH
Asks Kermit to apply patches to its memory image from the text file `MSKERMIT.PCH`, which must be stored on a disk and in a directory in the current DOS PATH, and which must correspond with the current MS-DOS Kermit version number.

PAUSE [{*seconds, hh:mm:ss*}]
Sleeps for the specified number of seconds or until the specified time of day, which cannot be more than 12 hours from the current time. Default interval is one second. Examples:

```
MS-Kermit>pause
MS-Kermit>pause 3
MS-Kermit>pause 23:59:59
```

Turns on the DTR signal (see Glossary). PAUSE can be interrupted by typing any key, which causes failure.

POP
Synonym for END. See END.

PUSH
Invokes an MS-DOS command processor "underneath" Kermit. Type `EXIT` at the DOS prompt to return to Kermit. Example:

```
MS-Kermit>push
C>format a:
C>exit
MS-Kermit>
```

QUIT
Synonym for EXIT (see EXIT). Example:

```
MS-Kermit>q
```

R
Special abbreviation for RECEIVE:

```
MS-Kermit>r
```

READ *variable-name*
Reads a line of text (up to a carriage return / linefeed line terminator) into the specified variable (macro argument or permanent variable). The line terminator is discarded. The command fails if a READ file has not been opened or has reached end of file. Example:

```
read \%a
if error goto eof
echo \%a
```

RECEIVE [*alternate-name*]
Waits for files from the other Kermit, which must be given a SEND command. If *alternate-name* is given, renames the first arriving file to this name. *alternate-name* may be a device (such as PRN) or any combination of disk, directory, and filename. If *alternate-name* includes a disk and/or directory name but not a filename, all arriving files will be placed there under their own names. Examples:

```
MS-Kermit>receive
MS-Kermit>rec newname.txt
MS-Kermit>rec a:
MS-Kermit>r \chris\
MS-Kermit>r a:\chris\
```

REINPUT *n text*
Like INPUT, but rereads text that has already been stored in the input buffer. The number *n* is a timeout interval, but it is ignored. See INPUT.

REMOTE *command*
Prefix for commands to be sent to a remote Kermit server. (The REMOTE commands are listed on page 225).

REPLAY *filename*
Replays a session log through the terminal emulator to reproduce a terminal session. Kermit's terminal type must be set the same as it was during logging. Example:

```
MS-Kermit>set term tek
MS-Kermit>replay usa.tek
```

RUN *command* [*arguments*]
Invokes an MS-DOS command or program, with any command-line arguments that may be given. Examples:

```
MS-Kermit>run basic
MS-Kermit>run edlin autoexec.bat
MS-Kermit>run ren oldname.txt newname.txt
```

S
Special abbreviation for SEND:

```
MS-Kermit>s oofa.txt
```

SEND *filename* [*alternate-name*]
Sends files to a remote Kermit receiver or server. If *alternate-name* is given, the first file is sent using that name. If the SEND command is typed on a line by itself, you are prompted separately for the two filenames. Examples:

```
MS-Kermit>send oofa.txt
MS-Kermit>sen oofa.txt newname.txt
MS-Kermit>s *.txt
MS-Kermit>send
 Local source file: oofa.txt
 Remote destination file: newname.txt
```

SERVER [{*seconds*, *hh:mm:ss*}]
Puts MS-DOS Kermit into SERVER mode. The MS-DOS Kermit server honors the following requests, within the restrictions established by the ENABLE and DISABLE commands (see DISABLE, ENABLE, and REMOTE):

SEND	REMOTE CD	REMOTE LOGIN
GET	REMOTE DELETE	REMOTE MESSAGE
FINISH	REMOTE DIR	REMOTE SEND
BYE	REMOTE HELP	REMOTE SET
LOGOUT	REMOTE HOST	REMOTE SPACE
	REMOTE KERMIT SET	REMOTE TYPE

The MS-DOS Kermit server can be run indefinitely (until it receives a BYE or FINISH command), or for the specified number of seconds, or until the specified time (unless a BYE or FINISH command comes first and FIN is not DISABLEd). Examples:

```
MS-Kermit>server                 (Serve forever)
MS-Kermit>server 3600            (For one hour)
MS-Kermit>server 20:00:00        (Till 8:00 P.M.)
```

After giving the SERVER command, you can get back to the `MS-Kermit>` prompt by typing *Ctrl-C* or just the letter C (see also SET SERVER, ENABLE, and DISABLE).

SET *parameter value*
Sets various parameters (listed beginning on page 227).

SHOW *topic*
Displays settings or other information related to the given topic (see page 243).

SPACE [*disk*]
Reports the available disk space on the current or specified disk. Examples:

```
MS-Kermit>space
 234,712 bytes available on drive C:
MS-Kermit>space a:
 Drive A: is not ready
```

You may use the SPACE command in script programs to test whether a disk drive exists and is ready:

```
space b:
if failure fatal {Drive B: is not ready}
cd b:
```

STATUS
Shows values of all SET parameters. Example:

```
MS-Kermit>stat
```

STAY
(from DOS command line only) Stays within Kermit after the other commands have executed. Normally Kermit exits back to DOS automatically if invoked with command-line arguments. Example:

```
C>kermit connect, stay
```

STOP
In a command file or macro, returns directly to `MS-Kermit>` prompt level (or exits to DOS if Kermit was invoked with command line arguments). Example:

```
if alarm stop
```

TAKE *filespec*
Executes MS-DOS Kermit commands from the specified file. A TAKE file may contain any MS-DOS Kermit commands, including other TAKE commands. Example:

```
MS-Kermit>take vt300.ini
```

If no disk or directory is included in the *filespec*, Kermit looks first in the current disk and directory. If the file is not found there, Kermit searches the disks and directories specified in your DOS PATH variable (if you have one) in the order in which they are listed.

TRANSMIT *filespec* [*prompt*]
Sends the characters from the file to the host with no error checking, just as if you were typing them (but faster). The *prompt* is the character to wait for from the host before sending the next line. The default prompt is linefeed (`\10`) or the current handshake character, if any (SET HANDSHAKE). Use `\0` to tell MS-DOS Kermit not to wait for any prompt character at all. (See also SET TRANSMIT.) TRANSMIT example:

```
MS-Kermit>transmit oofa.txt
MS-Kermit>tr oofa.txt \0
```

TYPE *filespec*
Displays a local file on the screen. Example:

```
MS-Kermit>type \autoexec.bat
```

The Kermit TYPE command just runs the DOS TYPE command, so long files will fly by faster than you can read them unless you type *Ctrl-S* and *Ctrl-Q* to stop and resume the display. To pause automatically after each screenful, use the DOS MORE command. Example:

```
MS-Kermit>run more < \autoexec.bat
```

VERSION
Displays the MS-DOS Kermit program version number and release date. Example:

```
MS-Kermit>v
IBM-PC MS-Kermit: 3.11 1 July 1991
```

WAIT {*seconds, hh:mm:ss*} **[CD] [DSR] [CTS]**
Turns on the DTR signal. Waits the specified number of seconds or until the specified time of day for the given modem signal(s) to appear. If all the given signals do not appear within the allotted time, set the FAILURE flag. Default time to wait is one second. If no modem signals are specified, WAIT is equivalent to PAUSE. Examples:

```
MS-Kermit>wait
MS-Kermit>wait cd
MS-Kermit>wait 5 cd
MS-Kermit>wait 10 cd dsr
MS-Kermit>wait 03:21:43 cts
```

WRITE {FILE, PACKET, SESSION, SCREEN, TRANSACTION} [*text*]
Writes the *text* to the specified file or log (if open) or to the screen. The text may contain variables and backslash codes. A terminating carriage return and linefeed (CRLF) is not written unless you supply it. The WRITE FILE command is used with files opened by the OPEN WRITE and OPEN APPEND commands. Example:

```
write session Begin session log at \v(date) \v(time)\13\10
```

Note: The syntax of the WRITE command has changed since version 3.0. The 3.0 version required a keyword such as DATE or TIME between the destination and the text. The new version is more flexible because variables of all kinds can be placed anywhere in the text.

IF Commands

The general form of the IF command is:

IF *condition command*

If the *condition* is true, the *command* is executed. The word NOT can be placed before the IF condition to reverse its meaning:

IF NOT *condition command*

which means that the *command* is to be executed if the *condition* is *not* true.

Here are MS-DOS Kermit's IF commands:

IF ALARM *command*
The *command* is executed if the SET ALARM time has passed. Example:

```
set alarm 180  ; A 3-minute egg timer
run cls        ; clear screen
:loop
echo \v(time)
pause 1
if not alarm goto loop
echo \7Your egg is ready!\7\13
```

IF COUNT *command*
Subtracts one from COUNT; if COUNT is greater than zero (see SET COUNT), the *command* is executed. Example:

```
set count 5
run cls
:loop
echo \v(count)
pause 1
if count goto loop
echo \7Blast off!\7
```

IF DEFINED *name command*
The *command* is executed if the macro or variable with the given *name* is defined (see DEFINE). Example:

```
if defined \%1 assign \%a \%1
```

IF ERRORLEVEL *number command*
The *command* is executed if the value of the ERRORLEVEL variable matches or exceeds the given *number*. Example:

```
if errorlevel 13 stop
```

This is equivalent to `if > \v(errorlevel) 13 stop`, which is the preferred form for portability and clarity.

IF EQUAL *word1 word2 command*
The *command* is executed if the character string *word1* is the same as *word2*. *word1* and *word2* may be variables, but neither constant nor variable may contain spaces. Alphabetic case is treated according to SET INPUT CASE. Example:

```
set input case ignore
ask \%a Should I stop? (yes or no)
if equal \%a yes stop
```

IF LGT *word1 word2 command*
The *command* is executed if *word1* is lexically (alphabetically) greater than *word2*. Alphabetic case is treated according to SET INPUT CASE.

IF LLT *word1 word2 command*
The *command* is executed if *word1* is lexically (alphabetically) less than *word2*. Alphabetic case is treated according to SET INPUT CASE.

IF = *number1 number2 command*
The *command* is executed if *number1* is equal to *number2*. *number1* or *number2* may be numeric constants or variables with numeric values, including the built-in variables like `\v(count)`, as well as the special words ARGC, COUNT, ERRORLEVEL, KEYBOARD, and VERSION, which are treated as variables only in this context. IF = COUNT does not subtract 1 from COUNT, as IF COUNT does. Examples:

```
if = \%1 3 echo The value is 3
if = \v(argc) 2 goto ok
if not = \v(argc) 2 goto bad
if not = version 310 stop
```

IF > *number1 number2 command*
The *command* is executed if *number1* is greater than *number2*. *number1* or *number2* are as in IF =. Examples:

```
if > ARGC 3 echo Too many arguments!
if not > \%n 4 echo \%n is less than or equal to 4.
```

IF < *number1 number2 command*
The *command* is executed if *number1* is less than *number2*. *number1* or *number2* are as in IF =. Example:

```
if < COUNT 5 echo Still counting...\13
```

Remember: When COUNT is used with `IF =`, `IF <`, or `IF >`, its value is not changed (see IF COUNT).

IF EXIST *filename command*
The *command* is executed if the given file exists. Example:

```
if exist \autoexec.bat run ren \autoexec.bat \a.tmp
```

IF FAILURE *command*
The *command* is executed if the most recent command failed. For INPUT, the IF FAILURE command is executed only if SET INPUT TIMEOUT PROCEED is in effect. Example:

```
set input timout proceed
input 10 login:
if failure goto bad
echo Got login prompt!\13
goto good
:bad
echo Failed to get login prompt.\13
stop
:good
```

IF SUCCESS *command*
The *command* is executed if the most recent command succeeded. After INPUT, this command is executed only if the INPUT command succeeded or if SET INPUT TIMEOUT PROCEED is in effect. Example:

```
set input timout proceed
input 10 login:
if success goto good
echo Failed to get login prompt.\13
stop
:good
echo Got login prompt!\13
```

REMOTE Commands

The following commands can be used only when communicating with a remote Kermit server. Results, if any, are displayed on the screen or if *> filespec* is added to the end of the command, the results are written to the specified file or device. Any REMOTE command can have its results redirected in this way.

REMOTE CD [*directory* [*password*]]
(also REMOTE CWD) Changes current directory on the remote host. If *directory* not specified, changes to default directory. Examples:

```
MS-Kermit>remote cd /usr/michele
MS-Kermit>remote cd
```

REMOTE DELETE *filespec*
Deletes remote file(s). Example:

```
MS-Kermit>remote delete $disk1:[dave]*.tmp
```

REMOTE DIRECTORY [*filespec*]
Lists remote file(s). Examples:

```
MS-Kermit>remote directory
MS-Kermit>rem dir $disk1:[rose]oofa.*
MS-Kermit>remote directory > remote.dir
```

REMOTE HELP
Asks the server to send a list of the services it provides:

```
MS-Kermit>remo help
GET files  REMOTE CD [dir]     REMOTE DIRECTORY [files]
SEND files REMOTE SPACE [dir]  REMOTE HOST command
MAIL files REMOTE DELETE files REMOTE WHO [user]
BYE        REMOTE PRINT files  REMOTE TYPE files
FINISH     REMOTE HELP
```

If you send commands to the server that are not on the list, the server sends back an error message.

REMOTE HOST *command*
Sends a command to the remote host in its own command language, passed through the remote Kermit server, which sends the results back. The command must not be an interactive command. Examples:

```
MS-Kermit>rem host cp q names
MS-Kermit>rem host grep -i kermit *
```

If the remote host command needs to include >, add an additional `> CON` to the end:

```
MS-Kermit>remote host sort < a > b > con
```

REMOTE KERMIT *command*
Sends a command to the remote Kermit server in its own command language. Example:

```
MS-Kermit>rem kermit set file type binary
```

REMOTE LOGIN [*user password*]
Logs in to a remote Kermit server that supports this feature. Use braces for spaces within fields. If *user* and *password* are omitted from the command line, you are prompted for them. If the password is given on the REMOTE LOGIN command line, it will echo; typed at the prompt, it won't. Examples:

```
MS-Kermit>rem login vincent {secret password}
MS-Kermit>rem login
 Username: vincent
 Password: ______
 Account:
```

For the `Account`, type your account if one is required, or just press the Enter key.

REMOTE MESSAGE *text*
Sends a one-line message to be displayed on the remote Kermit server's screen. Example:

```
MS-Kermit>rem message Hello Henry!
```

REMOTE PRINT *filespec* [*options*]
Sends the local PC file or files to the remote Kermit and asks the remote Kermit to print it (or them) with the specified options, if any, which must be given in the syntax of the remote system's print command.

REMOTE SET *parameter value*
Tells the remote Kermit server to change one of its settings. Example:

```
MS-Kermit>rem set file type binary
```

REMOTE SPACE [*area*]
Shows available disk space on the remote host in the current device or directory or the one you specify. Examples:

```
MS-Kermit>rem space
MS-Kermit>rem space $disk1:[gary]
```

REMOTE TYPE *filespec*
Displays remote file(s) on your PC screen. Example:

```
MS-Kermit>rem type oofa.txt
```

REMOTE WHO [*user*]
Displays users who are logged in to the remote system or information about the specified user. Examples:

```
MS-Kermit>remote who
MS-Kermit>rem who annette
```

SET Commands

SET ALARM {*seconds, hh:mm:ss*}
In scripts, sets an alarm (for use with IF ALARM). Examples:

```
MS-Kermit>set alarm 10
MS-Kermit>set alarm 22:00:00
```

SET ATTRIBUTES {CHARACTER-SET, DATE, LENGTH, TYPE} {ON, OFF}
Enables or disables the use of specific attributes in file attribute packets. Attribute packets are used by the file sender to tell the receiver the file's size, date, type, and so forth. If Kermit refuses to transfer a file for reasons that you believe are unjustified, you can disable the use of a particular attribute like this:

```
MS-Kermit>set attribute date off
```

You can disable attribute processing completely like this:

```
MS-Kermit>set attr off
```

SET BAUD *number*
Synonym for SET SPEED. Example:

```
MS-Kermit>set baud 2400
```

SET BELL {ON, OFF}
Whether to beep at the end of a file transfer. Unless told otherwise, Kermit will beep. Example:

```
MS-Kermit>set bell off
```

SET BLOCK-CHECK-TYPE {1, 2, 3}
Level of error checking for file transfer. Type 1 is a 6-bit checksum, type 2 is a 12-bit checksum, type 3 is a 16-bit cyclic redundancy check (CRC). The higher the type, the more effective the error checking. Type 1 is used by default. Example:

```
MS-Kermit>set block 3
```

SET COM*n address* [*IRQ*]
Tells Kermit the address to use for serial communication port COM*n*, where *n* is 1, 2, 3, or 4. The address is usually given in hexadecimal notation. To enter a hexadecimal number, type a backslash (\) followed by the letter `x` and the hex number, for example `\x02e8`. You may optionally specify an interrupt request number to use for this device. If you do not, Kermit assumes IRQ 3 or 4 (it safely tests for each). If you *do* specify an IRQ number other than 3 or 4, you run the risk of interfering with other PC devices (for example, PC/XTs use IRQ 5 for their hard disks; PC/ATs and above typically use IRQ 14). The IRQ number can range from 2 to 15, but IRQs above 7 are not used on 8088- and 8086-based CPUs. Study the technical manuals for your PC and serial communications device carefully before using this command to make sure the address or IRQ you specify is not also used by a disk or other critical device on your PC.

SET COUNT *number*
In scripts, sets up a loop counter (for use with IF COUNT). Also sets the value of the `\v(count)` built-in variable. Example:

```
set count 3
:loop
echo \v(count). hello\13
if count goto loop
echo goodbye\13
```

SET DEBUG {ON, OFF, PACKETS, SESSION}
Displays PACKETS during file transfer; displays control and 8-bit characters specially during terminal SESSION. ON means both PACKETS and SESSION. OFF means no debugging. Examples:

```
MS-Kermit>set debug packets
MS-Kermit>set deb ses
MS-Kermit>set deb off
```

SET DEFAULT-DISK *disk-name*
Default disk drive for sending and receiving files. Equivalent to CD *disk-name*. Example:

```
MS-Kermit>set def a:
```

SET DELAY *seconds*
In remote mode, the number of seconds to pause after a SEND command before sending the file. Example:

```
MS-Kermit>set delay 5
MS-Kermit>send oofa.txt
```

SET DESTINATION {DISK, PRINTER, SCREEN}
Default destination device for incoming files. Examples:

```
MS-Kermit>set destination printer
MS-Kermit>set dest screen
```

SET DISPLAY
See SET FILE DISPLAY and SET TERMINAL BYTESIZE.

SET DUMP *filespec*
Specifies screen-copy (screen dump) filename for text screens, that is, the file to which a text screen is copied when you strike a key that invokes the `\Kdump` function. `KERMIT.SCN` is the default. Example:

```
MS-Kermit>set dump rick.scn
```

SET DUMP does not affect graphics screens.

SET DUPLEX {FULL, HALF}
FULL means remote echo and Xon/Xoff flow control; HALF means local echo and RTS/CTS line access control. The default is FULL. Example:

```
MS-Kermit>set dup h
```

SET EOF {CTRL-Z, NOCTRL-Z}
Method for determining or marking the end of a PC file during file transfer. NOCTRL-Z (the default) means the end of file is its last character. CTRL-Z means the end of a file is marked by a Ctrl-Z character, even if it is not the last character in the file. Example:

```
MS-Kermit>set eof ctrl-z
```

SET ERRORLEVEL *number*
Status code to be returned by Kermit upon exit, for use by DOS batch. Kermit normally sets its status code automatically according to the success or failure of its SEND, RECEIVE, GET, and REMOTE commands (see Table 17-1). Also sets the value of the `\v(errorlevel)` built-in variable. Example:

```
MS-Kermit>set err 3
```

SET ESCAPE *character*
Escape character for CONNECT, normally *`Ctrl-]`* (`\29`). The character may be typed literally or entered using backslash notation. Example:

```
MS-Kermit>set esc \28
```

SET FILE {CHARACTER-SET, COLLISION, DISPLAY, TYPE, WARNING}
Sets file-related parameters (examine them with SHOW FILE).

> **SET FILE CHARACTER-SET {CP437, CP850, CP860, CP863, CP865, CP866}**
> Tells MS-DOS Kermit which IBM code page to use when translating a text file during file transfer (see Table I-7); by default it is your current code page. When sending a file, the file character set is translated into the transfer character set (see SET TRANSFER) if the file type is TEXT, and when receiving the transfer character set is translated to the file character set. Example:
>
> ```
> MS-Kermit>set file char cp860
> ```

SET FILE COLLISION {RENAME, OVERWRITE, DISCARD}
Tells Kermit what to do when an incoming file has the same name as an existing file: rename the incoming file, overwrite the existing file, or discard the incoming file.

SET FILE DISPLAY {REGULAR, SERIAL, QUIET}
Selects the format of MS-DOS Kermit's file transfer display. QUIET means no file transfer display at all; REGULAR means a continuously updated screen form; and SERIAL is for use with hardcopy or Braille terminals or speech synthesizers.

SET FILE TYPE {BINARY, TEXT}
If the file type is BINARY, no translations are done during file transfer. If the file type is TEXT, Kermit translates between the current transfer and file character sets. The default file type is TEXT. Use SET FILE TYPE BINARY to transfer `.EXE` files, and application-specific files (databases, spreadsheets, and so on).

```
MS-Kermit>set file type bin
```

SET FILE WARNING {ON, OFF}
Replaced by SET FILE COLLISION. ON is the same as SET FILE COLLISION RENAME. OFF is the same as SET FILE COLLISION OVERWRITE.

SET FLOW-CONTROL {NONE, RTS/CTS, XON/XOFF}
Selects the full-duplex flow control method. Xon/Xoff is the default. RTS/CTS is only available with real COM1–COM4 serial port devices. Example:

```
MS-Kermit>set flo none
```

SET HANDSHAKE {XON, BELL, ESC, CR, LF, NONE, CODE *ascii-code***}**
Half-duplex line turnaround character. To be used during file transfer and with the TRANSMIT command. Examples:

```
MS-Kermit>set handshake xon
MS-Kermit>set handsh code 25
MS-Kermit>set ha none
```

SET INCOMPLETE {DISCARD, KEEP}
What to do with an incompletely received file. The default is DISCARD. Example:

```
MS-Kermit>set inc keep
```

SET INPUT *parameter value*
Various parameters for the INPUT script command:

SET INPUT CASE {IGNORE, OBSERVE}
Whether to ignore or observe alphabetic case when scanning arriving characters for INPUT and REINPUT text. Case is observed by default. Also applies to IF EQUAL, IF LGT, and IF LLT. Example:

```
MS-Kermit>set inp case ign
```

SET INPUT DEFAULT-TIMEOUT *seconds*
How many seconds to wait for the specified input if a timeout interval is not specified. The default interval is one second.Example:

```
MS-Kermit>set inp def 5
MS-Kermit>input login:
```

SET INPUT ECHO {ON, OFF}
Whether the INPUT command should display characters on the screen as it reads them. Normally, the characters are displayed. Example:

```
MS-Kermit>set inp e off
```

Note: The terminal emulator is not active during INPUT. If you have SET INPUT ECHO ON and want formatted screens to appear correct, you should have a DOS console driver such as `ANSI.SYS` loaded. If escape sequences arriving during INPUT commands cause problems with your console driver, SET INPUT ECHO OFF.

SET INPUT TIMEOUT-ACTION {PROCEED, QUIT}
Whether MS-DOS Kermit should proceed to the next statement in a macro or command file if an INPUT command fails to read the specified characters, or else quit from the command file or macro. By default, Kermit proceeds. Example:

```
MS-Kermit>set inp tim quit
```

SET KEY [*keycode definition*]
Specify key redefinitions or keystroke macros (see page 238).

SET LOCAL-ECHO {ON, OFF}
Specifies whether MS-DOS Kermit should echo characters itself during terminal emulation (ON) or let the remote host echo them (OFF). SET LOCAL-ECHO ON is implied by SET DUPLEX HALF, and SET LOCAL-ECHO OFF is implied by SET DUPLEX FULL. The default is OFF, for full-duplex remote echoing. Example:

```
MS-Kermit>set loc on
```

SET DUPLEX HALF also enables RTS/CTS half-duplex line turnaround. If you need local echo, but don't want RTS/CTS, use SET LOCAL-ECHO ON instead of SET DUPLEX HALF.

SET LOG
Synonym for LOG (See LOG). Example:

```
MS-Kermit>set log packets p.log
```

SET MODE-LINE {ON, OFF}
Whether to display a mode line at the bottom of the screen during terminal emulation. Normally, the mode line is displayed. Example:

```
MS-Kermit>set mode off
```

SET NETBIOS-NAME *name*
Establishes your PC's NETBIOS network node name, so other Kermit programs on the same NETBIOS network can establish sessions with your PC. Kermit automatically appends a period and an uppercase letter K to the name you have given. Alphabetic case is significant in NETBIOS node names.

SET PARITY {NONE, EVEN, ODD, MARK, SPACE}
Character parity to use during terminal emulation and file transfer. If NONE, 8-bit data can be transferred; otherwise, only 7-bit characters can be used during terminal emulation, and a special prefixing mechanism is used for 8-bit data. Example:

```
MS-Kermit>set par even
```

SET PORT {*port, network, etc.*}
Selects a port, driver, or network for communication. The default port is COM1.

SET PORT COM1
(or COM2, COM3, or COM4) Selects a regular IBM or IBM-compatible communications port or internal modem (8250 or 16550A UART).

SET PORT 1
(or 2, 3, or 4) Equivalent to SET PORT COM1, COM2, COM3, or COM4. Example:

```
MS-Kermit>set port 2
```

The following SET PORT commands apply to local area network connections, which are explained in greater detail in Chapter 16 on page 189:

SET PORT BIOS1
(or BIOS2, BIOS3, or BIOS4) Selects communications port 1, 2, 3, or 4 but goes through the system BIOS Interrupt 14 driver rather than using Kermit's built-in UART driver. This allows use of Kermit through any kind of communications device that is supported by a BIOS-level COM driver. Examples include nonstandard internal modems and certain local area networks. Example:

```
MS-Kermit>set port bios1
```

SET PORT 3COM(BAPI)
Selects BAPI protocol for communication with asynchronous communication servers for use on 3Com and similar networks.

SET PORT DECNET [*host* [*password*]]
DECnet/DOS users can use this command to communicate with a VAX/VMS or other DECnet host. If you leave out the host name, Kermit uses the one from the previous SET PORT DECNET command, if any. Examples:

```
MS-Kermit>set port dec vaxine      (Normal)
MS-Kermit>set port dec cumin xyzzy (LAT password required)
```

SET PORT EBIOS {1, 2, 3, 4} [*name*]
Use the Extended Bios for communications with an EBIOS-based asynchronous communications server, such as IBM's LAN Asynchronous Connection Server versions 1.01 or 2.0 (LANACS) or AT&T's Asynchronous Gateway. Specify which serial port to emulate, 1 through 4, and the name of the desired port or hunt group on the server. Omitting the name resumes a previous session. SET SPEED and SET PARITY after the SET PORT command to ensure these parameters are sent correctly to the server (see Chapter 16).

SET PORT NETBIOS [*name*]
For direct station-to-station communication on NETBIOS networks without going through a file server. See also SET NETBIOS-NAME. Example:

```
MS-Kermit>set port netbios         (Set self up as server)
MS-Kermit>set port netbios lisa.K  (Connect to server)
```

SET PORT NOVELL
For accessing Novell asynchronous communication servers using NASI / NACS protocols. Example:

```
MS-Kermit>set port nov
```

After you CONNECT, you will have a dialog with the Novell server to select the desired service.

SET PORT OPENNET [*name*]
For Intel OpenNET. Works like SET PORT NETBIOS. Example:

```
MS-Kermit>set port opennet connie.K
```

SET PORT TCP/IP *host*
Connect to an Internet TCP/IP host using Kermit's built-in TCP/IP support.

SET PORT TELAPI *internet-address*
Connect to an Internet TCP/IP host through Novell LAN Workplace for DOS.

SET PORT TES *hostname*
Connect to a VAX/VMS system that is running NetWare/VMS. The PC must have Novell or InterConnections Inc. TES loaded.

SET PORT UB-NET1
For Ungermann-Bass Net/One. Example:

```
MS-Kermit>set port ub
```

SET PRINTER *name*
Redirects printer output that is initiated during terminal emulation by pressing PrintScreen or Ctrl-PrintScreen or by host-generated escape sequences to the specified file or device. The *name* can be a filename or a device name such as NUL.

SET PROMPT *text*
Changes the MS-Kermit> prompt to *text*. Example:

```
MS-Kermit>set prompt Jeannette>
Jeannette>
```

Variables in the SET PROMPT text are evaluated at the time the SET PROMPT command is given:

```
MS-Kermit>set prompt \v(dir)-\v(time)>
C:\ROBERTA-13:45:23>cd \sal
C:\ROBERTA-13:45:23>
```

So the current directory and time of day do not chance with each prompt.

SET RECEIVE *parameter value*
Requests the remote Kermit to use the specified parameters (listed on page 237).

SET RETRY *number*
Packet retransmission threshold. Normally, Kermit will try to send a particular packet up to five times before giving up. Use this command to raise or lower this number. Example:

```
MS-Kermit>set ret 20
```

SET SEND *parameter value*
Use the specified parameters during file transfer (listed on page 237).

SET SERVER LOGIN *user password*
Establishes a username and password that must be sent by a REMOTE LOGIN command before the MS-DOS Kermit server will respond to any other requests. Example:

```
MS-Kermit>set server login milly xyzzy
```

SET SERVER TIMEOUT *seconds*
How often the MS-DOS Kermit server times out between commands, normally zero, meaning no timeout at all between commands. Example:

```
MS-Kermit>set serv tim 30
```

SET SPEED *number*
Specifies the serial communications port or EBIOS connection transmission speed in bits per second. Common values are 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, and 57600. Type SET SPEED ? for a complete list. Example:

```
MS-Kermit>set sp 9600
```

SET TAKE-ECHO {ON, OFF}
Specifies whether commands from TAKE files and macros are echoed on your screen during execution. Normally OFF. Example:

```
MS-Kermit>set tak on
```

SET TCP/IP *parameter value*
Set TCP/IP network parameters for use with SET PORT TCP/IP connections. Parameters include: ADDRESS (your PC's IP address), DOMAIN (your organization's IP domain name), GATEWAY (the IP address of your local network's gateway), SUBNETMASK (your local subnetwork's address mask), PRIMARY-NAMESERVER (the IP address of your primary nameserver), and SECONDARY-NAMESERVER (the IP address of your secondary nameserver). IP addresses are of the form `nnn.nnn.nnn.nnn` where each `nnn` is a decimal number, 0-255 (see Chapter 16).

SET TERMINAL *parameter value*
Terminal emulation parameters (listed on page 239).

SET TIMER {ON, OFF}
Enable or disable timeouts and automatic packet retransmission during file transfer. Normally ON. Example:

```
MS-Kermit>set tim off
```

SET TRANSFER CHARACTER-SET {TRANSPARENT, LATIN1, CYRILLIC}
Specifies the character set to be used for file transfer. LATIN1 means to translate between the current file character set (see SET FILE) and ISO Latin Alphabet 1. Cyrillic means to translate between the file character set and ISO Latin/Cyrillic. TRANSPARENT means not to translate characters at all. The default is TRANSPARENT. Example:

```
MS-Kermit>set transf char latin1
```

SET TRANSLATION INPUT *host-char screen-char*
Translates the specified arriving host character to the specified screen character during terminal emulation. SET TRANSLATION INPUT {ON, OFF} enables or disables translations entered by this method. They are normally disabled. Examples:

```
MS-Kermit>set transl inp \91 \132
MS-Kermit>set transl inp on
```

If the terminal character-set is TRANSPARENT, the host character is translated directly. Otherwise, the host character is translated into the current code page before SET TRANSLATE INPUT sees it, so the host character must be specified as a value in the current code page. SET TRANSLATE INPUT OFF leaves the translation table available, but disables its use until the next SET TRANSLATE INPUT ON command.

SET TRANSLATION KEYBOARD {ON, OFF}
Enables/disables character-set translation for the international characters you type during CONNECT mode. This command affects only keys that have not been redefined with SET KEY and which do not have Kermit keyboard verbs assigned to them by default. Example:

```
MS-Kermit>set transl key off
```

SET TRANSMIT *parameter value*
Controls the behavior of the TRANSMIT command (see TRANSMIT). Normally, TRANSMIT sends a text file a line at a time and strips the linefeed (LF) from the end of the line, sending only the carriage return (CR, same as Enter), just as you would type it, and waits for the host to echo a linefeed before sending the next line. (The echoed LF is called the *prompt*.) Blank lines are sent as adjacent carriage returns. The following commands can be used to change the normal behavior:

SET TRANSMIT FILL-EMPTY-LINE {NONE, SPACE, *character***}**
Normally, a blank line is sent as a single CR. Some hosts or text editors treat blank lines as end of file. You can have Kermit add a character to each blank line to make such hosts or editors accept them. Examples:

```
MS-Kermit>set transm fill space
MS-Kermit>set transm fill X
```

SET TRANSMIT LINE-FEEDS-SENT {ON, OFF}
Tells Kermit to send both the CR and the LF at the end of each line, rather than just the CR:

```
MS-Kermit>set transm line on
```

SET TRANSMIT PROMPT *char*
Changes the default prompt for TRANSMIT from linefeed to whatever your host application is using. For example, if you are transmitting into a text editor whose prompt ends with a question mark:

```
MS-Kermit>set transm prompt \63
```

Use SET TRANSMIT PROMPT `\0` to tell Kermit not to wait for any prompt at all.

SET UNKNOWN-CHARACTER-SET {KEEP, CANCEL}
Tells what to do when a file arrives whose transfer character-set (as announced in the accompanying attribute packet) is unknown to MS-DOS Kermit: keep the file (without translating its characters) or cancel the file transfer. The default is KEEP.

SET WARNING {ON, OFF}
Obsolete. See SET FILE COLLISION. SET WARNING ON is the same as SET FILE COLLISION RENAME and SET WARNING OFF is the same as SET FILE COLLISION OVERWRITE.

SET WINDOWS *number*
Specifies packet window size, 1 to 31, for use only on full-duplex connections. Improves speed of file transfer over long-distance connections. Example:

```
MS-Kermit>set win 6
```

MS-DOS Kermit's total packet storage space is about 2000 characters, so the window size times the packet length should not exceed this number. If it does, Kermit will adjust automatically.

SET SEND and SET RECEIVE Commands

The SET RECEIVE commands tell MS-DOS Kermit to tell the other Kermit what parameters to use during file transfer. The SET SEND commands tell MS-DOS Kermit to use the given parameters when sending packets, even if the other Kermit asks for something else:

SET SEND DOUBLE *char*
Specify the ASCII code of a character to be doubled in outbound packets.

SET {SEND, RECEIVE} END-OF-PACKET *char*
Packet terminator to use, normally carriage return (`\13`). Change this only if carriage return does not work. Example:

```
MS-Kermit>set rec end \27
```

SET RECEIVE IGNORE-CHAR *char*
ASCII value of character to be ignored (discarded) in incoming packets.

SET {SEND, RECEIVE} PACKET-LENGTH *number*
Maximum packet length. SET REC PACK 94 or greater enables long packets; SET SEND PACK *xx* overrides the negotiated length, but only if *xx* is shorter. Longer packets will speed up file transfer if the connection is not noisy. MS-DOS Kermit's maximum length is 2000. Example:

```
MS-Kermit>set rec pack 1000
```

SET {SEND, RECEIVE} PADCHAR *character*
Prepacket padding character to use. Rarely needed. Example:

```
MS-Kermit>set send padc \127
```

SET {SEND, RECEIVE} PADDING *number*
Number of padding characters to send (SET SEND) or to request (SET RECEIVE) per packet, normally zero. Rarely needed. Example:

```
MS-Kermit>set rec padd 3
```

SET SEND PAUSE *number*
Interpacket pause in milliseconds (thousandths of seconds). Only for sending. Example:

```
MS-Kermit>set send pau 100
```

SET {SEND, RECEIVE} QUOTE *character*
Control-character prefix to use when sending packets. Normally #. Should never need to be changed.

SET {SEND, RECEIVE} START-OF-PACKET *character*
Control character that marks the beginning of a packet. Normally Ctrl-A (`\1`). Change this if the Ctrl-A character is intercepted by some device (like a modem) between your PC

and the other computer. You must change the start-of-packet character in *both* places. In this example, the packets that MS-DOS Kermit sends are changed to begin with a Ctrl-B character. The packets that the other Kermit (C-Kermit in this case) sends will still start with Ctrl-A.

```
MS-Kermit>connect
C-Kermit>set rec start 2
C-Kermit>server
Alt-X
MS-Kermit>set send start 2
MS-Kermit>send oofa.txt
```

SET {SEND, RECEIVE} TIMEOUT *number*
Timeout interval, in seconds, while waiting for a packet to arrive. SET RECEIVE TIMEOUT tells MS-DOS Kermit to tell the other Kermit to set its timeout to the given number. SET SEND TIMEOUT sets MS-DOS Kermit's own timer, overriding whatever the other Kermit might request. A value of zero (0) means to turn the timer off altogether. Example:

```
MS-Kermit>set rec tim 3
```

The SET KEY Command

The SET KEY command tells Kermit to transmit a certain character or sequence of characters or to perform a certain function when a given key is pressed during terminal emulation. Terminate the SET KEY command by pressing the Enter key. Kermit prompts you for a key to be pressed and then for a new definition:

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

The key definition can be any of the following:

- A single character, like `X`.
- A backslash code representing a single character, like `\127`.
- A character string, like `Hello there!`.
- A character string containing backslash codes, like `\7Help!\13`.
- A Kermit verb, like `\kexit`.
- A temporary or permanent variable, like `\%1` or `\%a`. Variables in key definitions are evaluated at the time the definition is made, not at the time the key is pressed.
- An invocation of a user-defined macro called *name*, `{\K`*name*`}`.

- A character string containing any combination of Kermit verbs, backslash codes, variables, and macro invocations.
- An empty definition (just press Enter) to remove the key's current definition and restore its default definition.
- *Ctrl-C* to cancel the SET KEY command and preserve the key's old definition.
- A question mark to show the kinds of definitions available.

A key definition can also be entered on one line by including the key's scan code followed by its definition:

```
SET KEY \315 login\13
```

This assigns the string `login`, followed by a carriage return (`\13`) to the F1 key, whose scan code is `\315` (see Table I-9).

SET KEY LK tells Kermit that your PC has an LK250 (DEC) keyboard with a driver loaded for it. SET KEY CLEAR removes all definitions and restores the built-in default set. SET KEY OFF means use DOS rather than the BIOS to obtain keystroke scan codes; SET KEY ON means use the system BIOS.

The SET TERMINAL Command

The following commands control many aspects of terminal emulation. To examine their current settings, use the SHOW TERMINAL command.

SET TERMINAL ARROW-KEYS {APPLICATION, CURSOR}
Explicitly puts the arrow keys (or more precisely, the Kermit verbs associated with them; see Table II-2) in application or cursor mode.

SET TERMINAL BELL {AUDIBLE, VISUAL, NONE}
Controls whether arriving BEL characters (ASCII character 7) ring the PC's bell[19] (beep) or flash the screen. AUDIBLE is the default. NONE discards incoming bell characters.

SET TERMINAL BYTESIZE {7, 8}
Tells whether to discard the 8th bit of incoming characters. 7 is the default, meaning to discard the 8th bit, even if PARITY is set to NONE. Use 8 if you are certain that parity is not in use and you need to use 8-bit character sets during terminal emulation. Synonym: SET [TERMINAL] DISPLAY {7,8}.

[19]The noise made by a terminal is called a bell because the earliest terminals, Teletypes (vintage 1930–1970), actually had bells. Modern terminals usually beep.

SET TERMINAL CHARACTER-SET *name* **[G0 [G1 [G2 [G3]]]]**
Selects the terminal character set and optionally invokes it into one or more of the specified graphic character tables (see page 289 in Appendix II). The ability to use these character sets depends on which kind of terminal Kermit is emulating; only the VT220 and VT320 have international characters. 7-bit character set names include ASCII, BRITISH, DEC-SPECIAL, DUTCH, FINNISH, FRENCH, FR-CANADIAN, GERMAN, ITALIAN, NORWEGIAN/DANISH, PORTUGUESE, SPANISH, SWEDISH, and SWISS. Names of 8-bit character sets are LATIN1 (the default), DEC-MCS (DEC Multinational Character Set), TRANSPARENT (the current IBM code page), and ALTERNATE-ROM. Examples:

```
MS-Kermit>set term vt320
MS-Kermit>set term char italian
MS-Kermit>set term char latin1 g1
MS-Kermit>set term char dec-mcs g1 g2 g3
```

SET TERMINAL CLEAR-SCREEN
Clears the screen:

```
MS-Kermit>set term clear
```

SET TERMINAL COLOR *number* [, *number* [, *number*]]
Sets the foreground and background colors for text terminal emulation. Use SET TERMINAL GRAPHICS COLOR to set the colors of the graphics screen. The *numbers* are:

0	For no snow on IBM Color Graphics Adapter (CGA)
1	High-intensity foreground
10	For fast screen update on IBM EGA and above
3*x*	Foreground color
4*x*	Background color

where *x* is the sum of any of 1 (Red), 2 (Green), and 4 (Blue). Examples:

```
MS-Kermit>set term color 34, 47
MS-Kermit>set term color 1, 31, 45
```

SET TERMINAL CONTROLS {7-BIT, 8-BIT}
When emulating a VT220 or VT320, tells whether to send 7-bit or 8-bit control sequences when DEC function, arrow, editing, or keypad keys are pressed. (See Appendix II.) The default is 7-BIT. Example:

```
MS-Kermit>set term cont 8
```

SET TERMINAL CURSOR-STYLE {BLOCK, UNDERLINE}
Selects the cursor style. The default is underline. Example:

```
MS-Kermit>set term curs b
```

SET TERMINAL DISPLAY {7, 8}
Same as SET TERMINAL BYTESIZE.

SET TERMINAL DIRECTION {LEFT-TO-RIGHT, RIGHT-TO-LEFT}
Chooses the direction in which characters are written on the screen during terminal emulation. Normally left to right. Use right to left for Hebrew, Arabic, or just for fun:

```
MS-Kermit>set term dir right
```

SET TERMINAL GRAPHICS *name*
Specifies the type of graphics adapter in your PC: CGA, EGA, VGA, and others, for purposes of Tektronix emulation. MS-DOS Kermit automatically guesses what kind of adapter you have. Use this command if it guessed incorrectly. Type question mark to find out which graphic adapters are supported. Example:

```
MS-Kermit>set term gr cga
```

SET TERMINAL GRAPHICS CHARACTER-WRITING {OPAQUE, TRANSPARENT}
Tells whether text characters written on graphics screens should be transparent (let graphics show through) or opaque (remove underlying graphics).

SET TERMINAL GRAPHICS COLOR *number* [, *number* [, *number*]]
Sets the colors for graphics screens. Numbers are as for SET TERMINAL COLOR.

SET TERMINAL GRAPHICS CURSOR {ON, OFF}
Tells whether there should be a text cursor during graphics terminal emulation. Default is ON, even though a real Tektronix terminal does not have a cursor.

SET TERMINAL KEYCLICK {ON, OFF}
On keyboards that support this, turns keyclick on or off. Example:

```
MS-Kermit>set term keyc off
```

SET TERMINAL KEYPAD {APPLICATION, NUMERIC}
Puts the numeric keypad into the specified mode. NUMERIC means send the digits or punctuation marks on the top of the key label; APPLICATION means send the DEC terminal keypad escape sequences associated with the bottom of the key label. Affects the action of the keyboard verbs associated with the DEC keypad (see Table I-4). Example:

```
MS-Kermit>set term keyp appl
```

SET TERMINAL MARGIN-BELL {ON, OFF}
Whether to ring the bell (or flash if SET TERM BELL VISIBLE) when the cursor nears the right (or left if SET TERM DIR LEFT) screen margin.

SET TERMINAL NEWLINE {ON, OFF}
ON means to send both a carriage return and a linefeed when you press Enter. OFF means to send only a carriage return (this is the default). ON is useful when two PC users are chatting with each other in Kermit CONNECT mode:

```
MS-Kermit>set term newl on
```

SET TERMINAL REPLAY (Synonym for REPLAY)

SET TERMINAL ROLLBACK {ON, OFF}
ON means to restore rolled-back screens to the end when new characters arrive. OFF means to display new characters at the current cursor position, even if it is in a rolled-back screen. OFF is the default. Example:

```
MS-Kermit>set term roll on
```

SET TERMINAL SCREEN-BACKGROUND {NORMAL, REVERSE}
REVERSE changes the foreground color to background, and vice versa. Example:

```
MS-Kermit>set term scr r
```

SET TERMINAL TABSTOPS {AT *n*, CLEAR AT *n*, CLEAR ALL}
Sets or clears screen tab stops at the specified positions. *n* can be a single number, a list of numbers, or *position*:*interval* to set tabs beginning at the specified *position*, every *interval* spaces. By default, tabs are set every eight spaces. Examples:

```
MS-Kermit>set term tab at 1, 2, 4, 8, 16, 32, 64
MS-Kermit>set term tab at 1:8
MS-Kermit>set term tab clear at 8, 32
MS-Kermit>set term tab clear all
```

SET TERMINAL TEK {ENABLE, DISABLE}
Tells MS-DOS Kermit whether it should automatically enter Tektronix graphics mode upon receipt of special escape sequences from the host (see Table II-24). This feature is ENABLEd by default. Example:

```
MS-Kermit>set term tek dis
```

SET TERMINAL TYPE {NONE, VT52, HONEYWELL, HEATH-19, VT100, VT102, VT220, VT320, TEK4010}
Selects the type of terminal to emulate. VT320 is the default. Example:

```
MS-Kermit>set term heath
```

SET TERMINAL UPSS { DEC-MCS, LATIN1 }
Sets the VT220/VT320 User Preferred Supplemental character Set (UPSS). The default UPSS is the DEC Multinational Character Set. The UPSS is invoked by certain host-generated escape sequences. (See Table II-11.)

SET TERMINAL WIDTH {80, 132}
Tells Kermit to put the video adapter into 80-column or 132-column mode. Takes effect the next time you give a CONNECT command. If Kermit can switch modes itself, it will do so automatically. Otherwise it will run the DOS batch program `COLS80.BAT` or `COLS132.BAT` (from the current disk or DOS PATH). You must fill in these files with whatever external DOS commands you have at your disposal to change the screen mode.

SET TERMINAL WRAP {ON, OFF}

Many host computers automatically break long lines into a series of lines that fit on your screen. Kermit assumes that the host does this, and so the default is OFF. If your host does not wrap long lines itself, the extra characters will "fall off" the right edge of your screen (or left edge, depending on SET TERM DIRECTION). To fix this:

```
MS-Kermit>set term wrap on
```

SHOW Commands

SHOW COMMUNICATIONS

Communication parameters: port, speed, parity, flow control, handshake, echo, and modem signals:

```
MS-Kermit>show comm

  Communications port: COM1       Speed: 9600
  Local echo: off                 Parity: none (8-bit data)
  Handshake used: none            Flow control: xon/xoff
  Duplex: full                    Display: Regular, 7-bit
  Debug: off

  Modem is ready:     DSR is on
  Carrier Detect:     CD  is on
  no Clear To Send:   CTS is off
```

SHOW FILE

File-related parameters: current path, destination, EOF mode, display, incomplete, warning, and take-echo:

```
MS-Kermit>sho file
  Path: C:\PAM              Discard incomplete file
  File destination: Disk    Warning (filename change): On
  EOF mode: NoCtrl-Z        Take-echo: Off
  Display: Regular, 7-bit   Attribute packets: On
```

SHOW KEY

Displays the definition of a selected key or all defined keys. You are asked to push a key, and Kermit shows you the definition:

```
MS-Kermit>sh key
Push key to be shown (? shows all): (F1 key is pressed)
 Scan Code \315 decimal is defined as
 Verb: Gold  \KGold
 Free space: 128 key & 128 string definitions, 1000 chars
```

SHOW LOGGING

Shows names and status of session, packet, transaction logs, and the screen dump file:

```
MS-Kermit>show log
```

SHOW MACROS [*name*]
Displays the name(s) and definitions of the given macro(s). Example:

```
MS-Kermit>sh mac ibm
 IBM = set timer on<cr>
  set parity mark<cr>
  set local-echo on<cr>
  set handshake xon<cr>
  set flow none<cr>
 Free space (bytes) for names: 993
```

The symbol <cr> means carriage return, which is what Kermit substitutes for the comma in your actual macro definition. If you don't give a macro name, Kermit shows all defined macros. You can also use SHOW MACRO to display the definition of a variable, which the ECHO command will not show you because ECHO fully evaluates its text before displaying it:

```
MS-Kermit>define \%b hello
MS-Kermit>define \%a \%b
MS-Kermit>echo \%a
hello
MS-Kermit>show macro \%a
\%A = \%b
```

SHOW MEMORY
Displays free memory:

```
MS-Kermit>sho mem
 DOS free memory (bytes): 251,024+48
 Total free bytes: 251,072
```

The first line shows the size of each free piece, and the second line shows the total size of all free pieces.

SHOW MODEM
Displays the status of the Carrier Detect (CD), Data Set Ready (DSR, meaning the modem), and Clear to Send (CTS) modem signals:

```
MS-Kermit>show modem
  Modem is ready:     DSR is on
  Carrier Detect:     CD  is on
  no Clear To Send:   CTS is off
```

SHOW PROTOCOL
Shows the values of the SET SEND, SET RECEIVE, and other file transfer protocol parameters:

```
MS-Kermit>show proto
```

In Kermit's display, ^A means Ctrl-A, ^M means Ctrl-M or carriage return, ^@ means ASCII character 0 (NUL), S: means a SET SEND parameter, and R: means a SET RECEIVE parameter.

SHOW SCRIPTS

Displays values of SET INPUT, SET ALARM, and SET COUNT parameters.

```
MS-Kermit>show scr
 Input echoing: on               Case sensitivity: Ignore
 Timeout (seconds): 1            Timeout-action: Proceed
 Alarm time: 00:00:00            INPUT-buffer-length: 128
 Errorlevel: 0                   Transmit fill-empty-line: none
 Transmit line-feeds-sent: off   Transmit prompt character: ^J
 Take/Macro COUNT: not active    Take-echo: off
 Take/Macro ARGC: not active     INPUT-BUFFER follows:
```

SHOW SERVER

Displays server-related parameters, including SET SERVER and ENABLE or DISABLE.

```
MS-Kermit>show serv
  Timeout (sec) waiting for a transaction: 0
  Login Username:
  Server commands available to remote user:
  CD/CWD: enabled       KERMIT: enabled
  DELETE: enabled       LOGIN:  enabled
  DIR:    enabled       MESSAGE:enabled
  FINISH: enabled       PRINT:  enabled
  GET:    enabled       SPACE:  enabled
  HOST:   enabled       TYPE:   enabled
```

SHOW STATISTICS

Displays statistical information on the most recent file transfer and values accumulated since starting Kermit.

```
MS-Kermit>sho stat
```

Kermit's file transfer efficiency is the file characters per second times 10 divided by the baud rate (COM and EBIOS ports only).

SHOW TERMINAL

Displays values of SET TERMINAL parameters. The bottom line shows the tab settings: T for each tab stop. The numbers mark every ten spaces.

```
MS-Kermit>sho term
```

SHOW TRANSLATION

Lists SET TRANSLATION INPUT settings:

```
MS-Kermit>set transl in \91 \123
MS-Kermit>set transl in \93 \125
MS-Kermit>sho on
MS-Kermit>sho transl
 Input Translation is on
 Translation table of rec'd bytes while in CONNECT mode -
 Format: [received byte (decimal) -> local byte (decimal)]
 [\91 -> \123]  [\93 -> \125]
```

SHOW VARIABLES

Lists MS-DOS Kermit's built-in variables and their values, for example:

```
MS-Kermit>show var
 \v(argc) = 0
 \v(count) = 0
 \v(date) = 05/12/1991
 \v(ndate) = 19910512
 \v(directory) = C:\TEMP
 \v(errorlevel) = 0
 \v(keyboard) = 101
 \v(platform) = IBM-PC
 \v(program) = MS-DOS_KERMIT
 \v(speed) = 9600
 \v(status) = 0
 \v(system) = MS-DOS
 \v(time) = 17:53:44
 \v(version) = 310
```

Glossary

Accent
A diacritical mark attached to a letter.

Acoustic Coupler
A modem having two rubber cups into which the telephone handset is inserted, or a pair of cups that can be connected to a direct-connect modem.

ACS
Asynchronous Communication Server. A device on a PC network that houses one or more serial ports that can be shared by all the PCs on the network. The ACS ports are typically connected to modems and telephone lines.

Acute Accent
A right-slanting accent mark, for example á.

Alt
The IBM PC key that you hold down while pressing another key in order to produce an Alt character. For example, `Alt-X` is produced by holding down Alt and pressing X.

Analog
Representation of computer data in some other form, like the kind of sound waves that are transmitted over telephone lines by modems. *Also see* Digital.

Answer
One of two modes a modem can be in. In answer mode, the modem waits for a call. *Also see* Originate.

Argument
In this book, a word of text that follows a DOS batch or Kermit macro invocation. Variables within the batch file or macro are replaced by the corresponding arguments.

ASCII
American Standard Code for Information Interchange, ANSI X3.4-1986. A standard 128-character code widely used by computers for representing and transmitting character data, in which each character corresponds to a number between 0 and 127. The ASCII character set is listed in Table I-5.

Asynchronous
Character- or byte-oriented serial data transmission, with delimitation of characters accomplished by start and stop bits. Used by PC serial ports and modems.

Asynchronous Adapter
The PC circuit board that controls the serial communication port, the device used by Kermit for connections to external modems and for direct connections to other computers. There are several kinds of asynchronous adapters. PC and PC/XT computers have a half-height card, called the Asynchronous Adapter, with one 25-pin male connector. PC/ATs may also use this card, or they may have the PC/AT Serial/Parallel Adapter with two connectors: a 9-pin male serial connector (the communication port) and a 25-pin female parallel connector (the printer port, which should not be used for communications). The PS/2 has a built-in 25-pin male connector for communications, not to be confused with the 25-pin female connector, which is the parallel printer port. Additional ports may be installed in the PS/2 using the Personal System/2 Dual Async Adapter/A, which has two male 9-pin connectors.

Autoanswer
A type of operation of a modem, in which it answers a telephone call automatically.

Autodial
A kind of modem that simulates a telephone's dialing mechanism, rotary or Touch-tone, in order to place a call, usually under computer control.

AUTOEXEC.BAT
A file which, when stored in the top-level directory of your PC's startup disk, is executed automatically when your PC is booted. The file must contain DOS commands or program invocations. Typical uses include starting TSRs (see TSR), setting your environment variables (see Environment), creating a RAM disk, establishing your DOS PATH, setting up your code page environment, prompting you to set the date and time, and choosing the format of your DOS prompt. See your DOS manual.

BAPI
3Com Bridge Applications Programmer Interface to 3Com networks, an asynchronous communication server (ACS) protocol used on PCs.

Batch
A feature of DOS that lets you collect DOS commands into a file and execute the file as if it were a program. DOS batch files have the filetype `.BAT`. Command line arguments can be referred to using the positional parameters `%1`, `%2`, . . . `%9`, and DOS environment variables can be used by surrounding their names with percent signs, e.g. `%PATH%`.

Baud rate
As most commonly used in PC communications, "baud" is the transmission speed expressed in bits per second (bps). For purposes of MS-DOS Kermit, baud and bps are synonymous, but this is not always true in other telecommunication areas.

BBS
Abbreviation for Bulletin Board System. A dialup computer service that typically lets you exchange messages with other users, read news on various topics, and upload and download software and files.

Binary
Referring to the number two. Binary notation is a way of writing numbers using only the two digits 0 and 1. Computers are made out of switches that have only two states, on (1) and off (0).

Binary File
A file containing codes that are used to control a device like a computer or printer. The contents of binary files usually depend on some particular hardware, and they should not be converted or translated in any way during transfer to another system.

BIOS
IBM's Basic Input Output System. The part of DOS that controls devices such as the disk, keyboard, serial port, and screen, and offers these control services to application programs.

Bit
A binary digit, 0 or 1.

Block Check
A quantity formed from a block (packet) of data by combining all its bytes, and then transmitting the result with the block itself so that the receiver of the block can determine whether it was corrupted in transit. Kermit supports three types of block checks: a one-character checksum (6 bits), a two-character checksum (12 bits), and a three-character CRC (16 bits). *Also see* Checksum, CRC.

Boot
Short for "bootstrap." The act of starting your PC. "Cold boot" means to start the PC by turning on the power. "Warm boot" means to restart the PC with the power already on by pressing the Ctrl, Alt, and Del keys simultaneously.

bps
Bits per second. Usually equivalent to baud. *See* Baud.

BREAK
A spacing condition on a communication line lasting about 0.275 seconds and generated by pressing `Alt-B` or `Ctrl-]B` on the PC keyboard during Kermit terminal emulation. Also, a "long BREAK" lasting about 1.5 seconds, produced by typing `Ctrl-]L`.

Buffer
A place to put arriving data until the intended recipient can get around to reading it, or a place to store outbound data until the transmitter gets around to sending it.

Byte
A unit of storage, abbreviated B, intended to hold a character, usually 8 bits long. Computer memory and disk capacity are often measured in thousands (K) or millions (M) of bytes (for example, 256KB).

Carriage Return
ASCII character number 13. This is the character that is produced when you press the PC's Enter key and that is used in conjunction with linefeed to terminate lines of text in IBM PC text files. Abbreviated CR.

Carrier
A continuous signal that is sent between two modems. The presence of carrier tells one modem that the other modem is in data transmission mode. The loss of carrier indicates the data connection is broken. An external modem usually has a carrier status light to let you know that it is communicating with the other modem.

CD
Carrier Detect. A signal to the PC from the modem indicating that it is connected to another modem. *Also see* Carrier.

Cedillia
A diacritical mark that looks like a small comma under a letter, for example Ç.

CGA
The IBM PC Color Graphics Adapter.

Character
A discrete unit of textual or control information, such as a letter, digit, or punctuation mark, belonging to a particular character set, like ASCII, IBM Code Page 437, or ISO Latin Alphabet 1.

Checksum
A block check based on the arithmetic sum of all the bytes in a block.

Circuit Board
A flat rectangular board containing electronic circuits that implement some component of a computer or communication device. Designed to be plugged into a slot, with signals passing through contacts on its edge. The asynchronous adapter is a circuit board, and so is an internal modem.

Circumflex
A diacritical mark, a "pointy hat" over a letter, for example ô.

Code
In data communications, the numeric or internal representation for a character in a particular character set, like ASCII or IBM Code Page 437.

Code Page
The name IBM uses for its character sets (see Table I-7).

Communication Port
A device that allows a computer or terminal to engage in data communication, appearing as an external connector on the back of a PC for a cable to a modem or another computer. *Also see* Asynchronous Adapter.

CONFIG.SYS
A file which, when placed in the top-level directory of your DOS startup disk, is used by DOS to locate and load device drivers into your PC's memory when the PC is started. `CONFIG.SYS` is also used to allocate memory for file handles and buffers, to specify your country information, and for other purposes. See your DOS manual.

Connector
A plug, of either male or female gender, that provides contacts for one or more wires within a cable and that mates with a similar plug of opposite gender to provide the desired electrical circuits. The connectors used most commonly with PCs for data communication are D-connectors (so called because they are shaped like the letter D) with either 25 pins (DB-25) or 9 pins (D-9). The DB-25 is often called an RS-232 or EIA connector.

Console
The primary input/output device with which a person controls a personal computer or a time-sharing session on a shared computer. The keyboard and screen.

Control Character
An ASCII chararacter in the range 0 to 31, plus ASCII character 127, contrasted with the printable, or graphic, characters in the range 32 to 126 (see Table I-5). Produced on an ASCII terminal by holding down the Ctrl key and typing the desired character. Standard 8-bit character sets such as ISO Latin-1 also have 32 additional control characters in the range 128 to 159.

CP
Abbreviation for Code Page.

CPU
Central processing unit. The part of the computer that executes instructions. The CPU of the original IBM PC, PC/XT, and many other personal computers is an Intel 8088. The Intel 80286 is used in the PC/AT and lower PS/2 models, the 80386 in higher PS/2 models and many non-IBM PCs and servers, and the 80486 is used in high-end PCs and servers. Variations (8086, 80186, 80C86, etc.) are also occasionally used. Other types of computers (mainframes, minicomputers, UNIX workstations) have completely different kinds of CPUs not necessarily manufactured by Intel.

CR
Carriage return (ASCII 13, Control-M).

CRC
Cyclic redundancy check. An error-checking technique in which a block of data is viewed as a long sequence of bits to be divided by a certain binary number, with the remainder used as the block check. *Also see* Block Check.

CRLF
Carriage return and linefeed, the sequence of ASCII characters (numbers 13 and 10) used by MS-DOS and by the Kermit protocol to delimit lines in a text file.

CTERM
The DECnet virtual terminal protocol used by the VAX/VMS SET HOST command.

Ctrl
Control. The key that you hold down while pressing another key (a letter or certain punctuation marks) in order to produce a control character. For example, `Ctrl-C` is produced by holding down Ctrl and pressing C.

CTS
Clear To Send. A signal used by a modem to regulate the flow of data from a terminal or computer. When the modem turns on CTS, the terminal is allowed to send data. When the modem turns off CTS, the terminal is not supposed to send data.

Cursor
The rectangular block or underscore on your CRT screen that indicates the current position.

Cyrillic
A family of alphabets used in the writing systems of Eastern European languages such as Russian, Ukrainian, Bielorussian, Serbian, Macedonian, etc., devised by the Slavic apostle Saint Cyril (827-869).

Data
Information as it is stored in, or transmitted by, a computer or terminal.

Dead Key
On certain IBM PC national keyboards, a key that acts as a prefix for another key to produce a special character. Press the dead key first, then the letter. For example, on the German keyboard, an apostrophe followed by e produces e-acute; apostrophe is the dead key.

Dedicated Line
A communication line that connects two devices with relative permanence, for example, a direct line from a terminal to a computer, or a leased telephone circuit. The opposite of a switched or dialup line.

Default
The value that is used for some parameter when no other value is explicitly provided. For example, the default Kermit block check is 1, and it will be used unless you explicitly tell Kermit to use type 2 or 3 by using the SET BLOCK-CHECK command. *Also see* Block Check.

Diaeresis
A diacritical mark, two dots over a letter, for example Ü.

Dialup
A data connection established with a telephone call, usually involving modems.

Digital
Representation of data by discrete 0s and 1s rather than continuous (analog) voltages. A PC is digital internally, and its communication port is digital. A modem converts digital computer data to analog waveforms (similar to speech) for transmission on telephone lines.

Direct-Connect Modem
A modem that connects directly to the telephone line via a modular jack.

Directory
A file on a disk that contains a list of other files with their physical locations on the disk, and possibly other information about them, such as size and creation date.

Directory Name
In MS-DOS, a sequence of characters that identifies a particular directory. The directory name is followed by a backslash character (\), for example, `PROGRAMS\`. Each disk can have many directories, and each directory can contain other directories. *Also see* Subdirectory.

Disk
A rotating magnetic storage medium for digital information, similar to a phonograph record, but possibly having more than one platter mounted on a central spindle. Disks are generally classified as hard (usually permanent, high capacity) and floppy (single platter, flexible, removable, moderate capacity). Floppy disks are also called diskettes.

Diskette
A single-platter, removable disk. Can be flexible, like an 8-inch or 5.5-inch floppy diskette, or rigid, like the 3.5-inch diskette used in the IBM PS/2.

DOS
Disk Operating System. A computer operating system that uses a magnetic disk as its principal medium of permanent storage. Also, short for MS-DOS and PC-DOS.

Download
To transfer a file from another computer to your PC.

DSR
Data Set Ready. A signal from a modem to the PC that says the modem is turned on and in data mode.

DTR
Data Terminal Ready. A signal from the PC to a modem that says the PC is turned on and ready to communicate. Some modems will refuse to communicate with your PC unless the PC is sending the DTR signal. MS-DOS Kermit starts sending the DTR signal as soon as you give the CONNECT, PAUSE, or WAIT command and keeps sending it until you give the HANGUP command.

Duplex
The degree to which a channel permits two-way traffic. Half duplex means traffic can go either way, but only one way at a time; full duplex means traffic can go both ways at the same time. *Also see* Echo, Full Duplex, Half Duplex.

EBCDIC
Extended Binary Coded Decimal Interchange Code. The character code used on IBM mainframes. Not covered by any formal standards but described definitively in the IBM System/370 Reference Summary.

Echo
How a character typed at a terminal, or a device emulating a terminal, is sent to the screen. Local echo means the terminal itself copies the character to the screen; usually associated with half-duplex communication. Remote echo means the system to which the character is transmitted sends it back to be displayed, possibly modified; this can be done only on full-duplex connections.

EGA
The IBM PC Enhanced Graphics Adapter.

Enter
The IBM PC key that terminates a command or a line of text. During terminal emulation, the Enter key sends a carriage return, ASCII character 13.

Environment
A place in your PC's memory for storing textual information that can be shared by all programs. *Environment* variables are established by the DOS SET command, and are accessed by various application programs, including Kermit.

ESC
Escape, ASCII character 27, Control–[.

Escape Character
A character used to get the attention of Kermit during CONNECT mode. Not to be confused with ASCII ESC. MS-DOS Kermit's default escape character is `Ctrl-]`.

Escape Sequence
A sequence of characters that selects a certain function. For instance, Kermit, during CONNECT, will accept a variety of escape sequences from the keyboard as commands. These consist of Kermit's escape character followed by a single character that selects the function, for example, `Ctrl-]B` to send a BREAK signal. At the same time, Kermit's terminal emulator responds to escape sequences sent from the host to accomplish screen formatting.

Ethernet
A local area network technology in which stations communicate with each other at 10 Mbps over a shared cable.

Even
See Parity.

External Modem
A modem that is not mounted internally in a PC. Usually portable, requiring its own power source or drawing power from the telephone line.

File
A collection of data that is stored on a disk and that has a name.

File Group
A collection of files that can be referred to using a single file specification that contains wildcard characters. *Also see* Wildcard.

Filename
In MS-DOS, a sequence of one to eight characters, followed by a period (.), followed by zero to three characters, used to identify a file within a directory, for example, `OOFA.TXT`.

File Specification
In MS-DOS, a sequence of characters composed of a disk letter followed by a colon (:), followed by a directory name enclosed in backslashes (\), followed by a filename, for example, `A:\PROGRAMS\OOFA.C`. If the disk is the same as the current disk, the disk designator can be omitted, for example, `\PROGRAMS\OOFA.C`. If the disk and directory are the current ones, both disk and directory designators can be omitted, for example, `OOFA.C`. If the disk is different but the directory is the same, the directory can be omitted, for example, `A:OOFA.C`. The filename portion of a file specification can include the wildcard characters `*` and `?` to denote a file group. *Also see* Wildcard.

Flow Control
The process by which the flow of data in the full-duplex communication environment is regulated so its arrival is coordinated with the capacity of the receiver to process it. MS-DOS Kermit uses Xon/Xoff (software) flow control by default. RTS/CTS (hardware) flow control is also available.

Front End
A communication processor for a host computer that operates independently from it but is closely tied to it. The front end relieves the host from the burden of detailed control of multiple terminals.

Full Duplex
A channel that permits simultaneous two-way data traffic between two devices. Full-duplex connections usually use Xon/Xoff or RTS/CTS flow control and remote echo. *Also see* Flow Control, Half Duplex, Echo.

Gateway
A device that connects two or more networks allowing traffic to flow between them, possibly reformatting and/or readdressing messages from one network type to another, and filtering messages that need not cross network boundaries.

Grave Accent
A left-slanting accent mark, for example ò.

Half Duplex
A channel that permits data transmission in both directions, but only in one direction at a time. Half-duplex connections usually use local echo and software line-turnaround handshake or hardware line access control like RTS/CTS.

Handshake
A method for granting permission to transmit a half-duplex channel. Usually the terminal or PC sends a carriage return as its handshake, and the host uses the Xon character (Ctrl-Q). Used by Kermit during file transfer over linemode connections to IBM mainframes.

Hex
Slang for hexadecimal.

Hexadecimal
Numeric notation in base 16, using the digits 0–9 and A–F to represent the numbers 0–15. For example, 9 hex is 9 decimal, hex A is 10 decimal, F hex is 15 decimal, and 10 hex is 16 decimal.

Host
A multiuser computer or time-sharing system to which a terminal (or a PC emulating a terminal) may be connected, such as a VAX computer or an IBM mainframe.

Input/Output
The process of getting data into and out of a computer, whether from a peripheral device like a disk or through a communication line to a terminal or another computer. Called I/O for short.

Interface
Computer jargon for something that allows two otherwise incompatible components to work together by satisfying their respective physical and logical requirements and making any necessary conversions of format, timing, voltage, etc. A connector is a kind of interface; so is the serial port. The aspect of a software program that interacts with a person is sometimes called the user interface. The console is said to be the user's interface to the system.

Internet
The worldwide TCP/IP network.

IP
Internet Protocol, the routing protocol and addressing conventions used in the Internet.

IRQ
Interrupt Request. PC devices generate interrupts when they need servicing, for example when data is ready to be input. Each active device is supposed to have a unique interrupt request (IRQ) number that identifies it to the operating system or application program. Kermit uses interrupts to read from the serial communication port, and therefore must know the IRQ number for the communication device. Kermit attempts to learn the IRQ number automatically, but you can give the command SET COM*n address IRQ* for nonstandard communication devices.

ISO
International Organization for Standardization. A voluntary international group of national standards organizations that issues standards in all areas, including computers, information processing, and character sets.

ISO Standard 8859
An ISO standard specifying a series of 8-bit computer character sets that include characters from many languages. These include the ISO Latin Alphabets 1–9, which cover most of the written languages based on Roman letters, plus special character sets for Cyrillic, Greek, Arabic, and Hebrew.

K
Abbreviation for kilo, meaning either 1000 or 1024.

LAN
Local area network.

LAT
Local Area Transport protocol, used by DEC Ethernet terminal servers.

Latin
Referring to the Latin, or Roman, alphabet, comprised of the letters A through Z, or to any alphabet based upon it.

Latin Alphabet
Any 8-bit character set based upon ISO Standard 8859. All such character sets include the Latin (Roman) alphabet. *See* ISO Standard 8859.

Leased Line
A permanent, dedicated communication line rented from the telephone company or another company.

Line
(1) A physical communication path, such as a telephone cable. (2) A sequence of characters in a text file intended to print on one line of a page or screen.

Linefeed
ASCII character 10. Used in conjunction with carriage return (ASCII 13) to delimit lines of text in an MS-DOS text file.

Local
Nearby, close to. When two computers or devices are connected, the local computer is the closer one. When two Kermit programs are connected, the local Kermit is the one that the user interacts with most directly (the one that has the CONNECT command).

Local Area Network
A data communication network that allows computing devices in a building or on a campus to communicate at high speeds.

Local Echo
Immediate display on the local screen, by the terminal or PC, of characters sent to a remote computer. Associated with half-duplex communication.

Long Packet
A Kermit packet whose length is greater than the normal maximum of 94. Long packets have a special format, in which an extended length field allows packets to be up to 9024 characters long. The maximum packet length for MS-DOS Kermit is 2000.

Loopback connector
A data connector that sends back to the computer whatever it receives.

M
Abbreviation for mega, meaning either one million or 1,048,576.

Mainframe
Commonly used to mean a big computer, as distinct from a minicomputer or microcomputer. In this book, it means any multiuser computer in which a user's console is also the user's only communication channel with the computer.

Mark
See Parity.

Medium
Something through which data is transmitted—copper wire, coaxial cable, optical fiber, empty space—or on which it is stored—magnetic disk, diskette, tape, CD ROM.

Memory
The internal, volatile, high-speed, solid-state storage of a computer, as distinguished from external, permanent, lower speed, rotating mechanical memories (for example disks, tapes) used for bulk storage.

Message
A unit of information, usually consisting of multiple bytes or characters, put into some specified format for transmission.

Microcomputer
In this book, any single-user computer whose console is distinct from its communication line.

Millisecond
One thousandth of a second.

MNP
Microcom Networking Protocol, used by modems for error correction and data compression.

Modem
A device (“modulator/demodulator”) that lets computers and/or terminals communicate over long distances using telephone connections by converting between serial digital data as output from a computer and analog waveforms suitable for transmission on a telephone line.

Modem Eliminator
See Null Modem.

Modem Signals
Signals transmitted from a modem to the PC, or vice versa, by which the modem and PC tell each other their status. The modem gives the CD (Carrier Detect), DSR (Data Set Ready), and CTS (Clear To Send) signals to the PC, and the PC gives the DTR (Data Terminal Ready) and RTS (Request To Send) signals to the modem. *Also see* CD, CTS, DSR, DTR, RTS, and Table I-1.

MS-DOS
Microsoft’s Disk Operating System for microcomputers based on the Intel 8086 family of CPU chips. *Also see* PC-DOS.

MSKERMIT.INI
A text file containing Kermit commands which, when stored in your current DOS directory or in any directory in your DOS path, is executed automatically by MS-DOS Kermit when it starts. Its purpose is to customize MS-DOS Kermit to your particular communications environment and other preferences.

MSKERMIT.PCH
A text file which, when stored in your current DOS directory or in any directory in your DOS path, is executed by MS-DOS Kermit when you give it the `PATCH` command. Its purpose is to install corrections to the program’s internal logic (that is, to fix bugs).

Multiplexer
A device that allows multiple devices to share a single communication medium, for example to connect 16 terminals to 16 ports on a computer over a single cable. Used in pairs, one at each end; the transmitter multiplexes, the receiver demultiplexes.

Name Server
A process running somewhere on a network that translates network host names into network addresses upon request from other processes on the network.

Network
A permanent arrangement that allows two or more computers or devices to communicate with each other conveniently and reliably at high speeds, over dedicated media, and that typically requires special hardware and operating-system level software.

Noise
Corruption of data during transmission.

NRC
National Replacement Character set. A 7-bit character set that is a variation of ISO 646 (ASCII) in which certain nonalphabetic graphic characters are replaced by special language-specific characters.

NUL
ASCII character number 0, as distinct from the number zero or the ASCII character digit 0 (ASCII 48). Also, the MS-DOS null device.

Null Modem
A pair of connectors, possibly with a length of cable between them, that allows two computers or terminals to be directly connected without intervening modems or multiplexers, and that supplies the required modem signals by means of cross-connections and jumpers.

Odd
See Parity.

Off
(1) Not in effect (said of an option). (2) Zero (said of a bit).

On
(1) In effect (said of an option). (2) One (said of a bit).

Operating System
The software program that controls a computer at its most basic level.

Originate
The mode of operation for a modem when it places a data call, as opposed to receiving one.

OS
Operating system.

OS/2
Operating System/2, a possible successor to MS-DOS and PC-DOS for the IBM PS/2 line of computers and compatibles.

Packet
A message that consists of fields whose locations and interpretation are agreed upon by the sending and receiving entities, to be transmitted as a whole, and that typically contains sequencing, error checking, and other control information as well as data.

Packet Driver
Software that controls a PC's network board, providing a uniform interface to all applications that want to use the network.

Parallel Port
A PC device used for connecting a parallel printer and that appears as a 25-hole female connector on the back of the PC. It should not be used for communication with modems or other computers.

Parameter
A symbolic value standing for, or to be replaced by, a real value. For example, in LOG SESSION *filename*, *filename* is a parameter to be replaced by the name of an actual file.

Parity
An error detection method in which one bit in each 8-bit byte is set aside to indicate some property of the remaining bits in a byte or word. Odd parity means the parity bit is set to make the overall number of 1 bits odd; even makes the overall number of 1 bits even. Mark parity means the parity bit is always set to 1; space parity means it's always set to zero. No parity means the bit that would otherwise be used for parity may be used for data, and 8-bit data may be transmitted.

PATH
A list of devices and directories that is searched by DOS to find programs that you run by typing their names at the DOS prompt. The PATH is established by the DOS PATH= command, usually in the `AUTOEXEC.BAT` file.

PBX
Private branch exchange. A telephone system that serves the internal needs of an organization and that provides connections to the external phone system. Often used for data transmission as well as voice within the organization. Some PBXs can be used for data transmission as well as voice within the organization. May be digital or analog.

PC
Personal computer. In this book, the term PC refers to the entire IBM PC and PS/2 families and compatibles. *Also see* Microcomputer.

PC-DOS
The version of MS-DOS distributed by IBM for use on its PC and PS/2 familes. *Also see* MS-DOS.

Port
See Communication Port.

Port Contention Unit
A device that allows multiple terminals to be connected to multiple computers, in which terminal ports contend for computer ports. Typically, the port contention unit engages in a dialog with the user, asking which computer the user wishes to connect to.

Protocol
In data communication, a set of rules and formats for exchanging messages, generally incorporating methods of sequencing, timing, and error detection and correction.

Protocol Converter
A device that converts between IBM mainframe 3270 block-mode terminal and ASCII character-mode terminal protocols.

Public Data Network
A network providing access, on a subscription basis, to widely scattered and diverse services. SprintNet, Tymnet, and Datapac are examples.

RAM
Random Access Memory. The memory used by your PC's application programs, device drivers, TSRs, and most of its operating system. Parts of RAM can also be set up as a RAM disk: a very fast and silent disk drive.

Relative Directory
In MS-DOS, a directory name that does not start with a slash and is assumed to be a subdirectory of the current directory.

Remote
Said of the more distant, or less directly accessed, of two connected computers. A remote Kermit is the one running on the computer that the local Kermit has connected to.

Retry
In this book, a second or subsequent attempt at transmitting a particular Kermit packet.

ROM
Read-only memory. High-speed internal memory containing permanently recorded information.

Roman
Referring to the Roman, or Latin, alphabet, comprised of the letters A through Z.

RS-232-C
An Electronic Industries Association (EIA) standard that gives the electrical and functional specification for serial binary digital data transmission. The most commonly used interface between terminals (or computers) and modems (or multiplexers). Updated recently to RS-232-D.

RTS
Request To Send. A signal used by a terminal or computer to regulate the flow of data from the modem. When the terminal turns on RTS, the modem is not supposed to send data. When the terminal turns off RTS, the modem is allowed to send data.

RTS/CTS
A form of full-duplex flow control or half-duplex line access control that uses the RTS and CTS modem signals. Unlike Xon/Xoff, this is generally not an end-to-end mechanism; rather, it works between the PC and the device it is directly connected to, such as a high-speed modem. *See* RTS and CTS.

Serial
A form of data communication in which a character's bits are sent in series, one after another. The dominant mode of data transmission over distances greater than a few feet.

Serial Port
See Asynchronous Adapter.

Server
A program or intelligent device that provides specified services to users, or clients, in response to requests, usually over a communication line or network. Kermit programs can be put into server mode, in which they accept commands only from other Kermit programs.

Shift-In/Shift-Out
A technique in which a pair of ASCII control characters (Ctrl-N and Ctrl-O) are used as shifts for transmitting 8-bit data on a 7-bit connection.

Sliding Windows
A feature of the Kermit file transfer protocol that allows packets to be sent in a continuous stream over a full-duplex connection. Up to 31 packets may be sent before an acknowledgment is required.

Space
(1) A kind of parity. *(2)* The representation of a binary zero or a BREAK condition on a serial communication line.

SprintNet
A commercial packet-switched wide area network in the USA run by US Sprint, formerly known as Telenet.

Subdirectory
In MS-DOS, a directory within another directory. In a file specification, each subdirectory name is preceded by a backslash (\) character, for example:

```
C:\PROGRAMS\SOURCE\OOFA.C
```

SuperKermit
A name used in the trade press and certain vendor literature to describe Kermit software that employs sliding windows during file transfer. *See* Sliding Windows.

Switched Line
A communication line subject to switching, like a dialed telephone connection.

TCP
Transmission Control Protocol. The transport layer of the TCP/IP protocol.

TCP/IP
A network protocol in widespread use for both local and wide area networking. The protocol of the worldwide Internet.

TDD
Telecommunication Devices for the Deaf. A Teletype or other hardcopy terminal with a built-in modem communicating at a very slow speed, using a special modulation technique and a limited 5-bit character code called Baudot.

Telecommunication
Serial data communication, possibly (but not necessarily) involving dialup telephone connections and modems.

Telenet
Former name of SprintNet. *See* SprintNet.

Telnet
A virtual terminal protocol used on TCP/IP networks. Not to be confused with Telenet.

Terminal
A device that allows a person to interact with a computer, with the person typing characters on a keyboard to send them to the computer, and with the computer's responses appearing on a screen or printer. May include the ability to interpret special character sequences to accomplish screen formatting. In general, differs from a computer by not having local permanent memory or general-purpose programmability.

Terminal Emulation
Behaving like a terminal. Said of software that runs on PCs or other computers that sends the user's keystrokes out the serial port and sends the port input to the screen. Sometimes includes the ability to interpret the same special sequences that a specific real terminal would obey. The Kermit CONNECT command performs terminal emulation.

Terminal Server
A network device that allows ordinary terminals with no networking capabilities of their own to participate in a network, provided hosts share a common protocol with the terminal server.

Text
Computer data intended for a person to read, or typed by a person, that consists of only printable characters and those control characters necessary for format control (carriage return, linefeed, tab, formfeed). Text files can be transferred between unlike systems and still remain useful. *Also see* Binary File.

Timeout
The process by which a program wakes up after waiting for some expected event (like input from a device) longer than a specified amount of time.

Translation Table
A list of all the translations from one character set into another.

TSR
Terminate and Stay Resident. The name for a class of PC programs that load themselves into memory and make themselves available for use by other programs or by you, in various ways. Examples include print spoolers, network shells, and pop-up hot-key utilities.

Tymnet
A public packet switched network service offered by Tymnet, Inc.

UART
Universal Asynchronous Receiver/Transmitter. An asynchronous communication port. MS-DOS Kermit for IBMs and compatibles provides high-speed interrupt-driven control of the 8250 and 16550A model UARTs, which are standard on IBM equipment and most clones, and in most internal modems.

Umlaut
The German word for diaeresis, a diacritical mark; two dots over a letter, for example Ö.

Unattended
Referring to an operation that can proceed automatically without human intervention.

UNIX
A popular operating system developed at AT&T Bell Laboratories and noted for its portability.

Upload
To transfer a file from your PC to another computer.

User
A person who is using a computer.

User Interface
The hardware and software with which a person communicates with a computer.

VAX/VMS
A proprietary operating system for DEC VAX computers.

Wildcard
A notation for referring to a group of files in a single file specification, by including pattern-matching characters. In MS-DOS, a * character in a filename matches any sequence of characters, and a ? character matches any single character.

Workstation
A single-user computer. Equivalent to a PC or microcomputer in that the console is separate from the communication line but is usually composed of more expensive components. Intended for more ambitious uses.

Xon/Xoff
The most common full-duplex flow control method, in which the receiver sends an Xoff character when its input buffer is close to filling up and an Xon when it has made room for more data to arrive. Also called "software flow control," in contrast to "hardware flow control" methods such at RTS/CTS.

Appendix I

Tables

Table I-1 shows the assignments of modem signals to pins for IBM PC 25-pin (DB25) connectors and IBM PC/AT and PS/2 9-pin (DB9) connectors.

Table I-1 RS-232-C Modem Signals and Pins

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

Hayes Smartmodem 2400 Commands

Table I-2 Selected Hayes Smartmodem 2400 Commands

Command	*Action*
AT	No action. Modem responds "OK" if it is in command state.
AT&C1	CD signal tracks carrier (recommended). AT&C0 keeps CD on always.
AT&D2	Modem hangs up and returns to command state if PC turns off DTR (recommended). AT&D0 makes modem ignore DTR signal from PC.
ATE1	Enable echoing of modem commands (recommended). ATE0 disables echoing.
ATM0	Turn off speaker. ATM1 turns on speaker while dialing.
ATQ0V1	Select verbal result codes (OK, CONNECT) rather than numeric.
ATX0	Enable OK, CONNECT, RING, and NO CARRIER result codes.
ATX1	Enable OK, CONNECT, RING, NO CARRIER, ERROR, CONNECT 1200, and CONNECT 2400 result codes.
ATX4	Enable OK, CONNECT, RING, NO CARRIER, ERROR, CONNECT 1200, NO DIALTONE, BUSY, CONNECT, and CONNECT 2400 result codes (factory setting).
AT&C1	Make CD track carrier (recommended). AT&C0 turns CD on always.
AT&D2	Make modem hang up phone and go back to command state if PC turns DTR off (recommended). AT&D0 makes modem ignore DTR.
ATDT*nnnnnnn*	Dial the phone number *nnnnnnn* (simulate Touch-Tone dialing). The phone number may contain digits, spaces, parentheses, and hyphens, which are ignored. A comma in the dial string causes the modem to pause. The letter W means wait for dial tone. An exclamation mark (!) means "hook flash"—hang up the phone for half a second, then reconnect.
ATDP*nnnnnnn*	Dial the phone number, like ATDT, but with pulse (rotary) dialing.
ATD*nnnnnnn*	Dial the phone number using the modem's default dialing method (Touch-Tone or Pulse).
ATH0	Hang up the phone.
+++	Return to command state without dropping the connection. This is the modem's "escape sequence." It is ignored unless a full second of "silence" precedes and follows it, to prevent consecutive plus signs in your data from interfering with communication.
ATO	(Letter O) Return to online state from command state.
ATZ	Initialize: Restore normal configuration.
ATS0=1	Enter answer mode (wait for a call).
ATS7=*nnn*	Wait up to *nnn* seconds for carrier. Default is 30 seconds.
ATS8=*nnn*	Duration of comma dial modifier. Default is 2 seconds.

MS-DOS Kermit CONNECT-Mode Escapes

Table I-3 lists the keyboard escape sequences available during terminal emulation. The *Character* column shows the key that you press after typing *Ctrl-]*; for example, C stands for *Ctrl-]C*. The *Verb* column shows the Kermit verb associated with this key, which you can use with the SET KEY command to assign this function to other keys. A complete list of keyboard verbs is given in Table I-4.

Table I-3 MS-DOS Kermit CONNECT-Mode Escapes

Character	*Verb*	*Description*
?	\Khelp	Help—prints the available single-character commands
0	\Knull	(the digit zero) Transmit a NUL (ASCII 0)
B	\Kbreak	Transmit a BREAK signal
C	\Kexit	Return to the MS-Kermit> prompt, connection stays open
F	\Kdump	File the current text screen in the screen dump file
H	\Khangup	Hang up the phone (or network connection)
L	\Klbreak	Transmit a long BREAK (1.8 seconds)
M	\Kmodeline	Toggle the mode line (i.e., turn it off if it is on and vice versa)
P	\Kdos	Push to DOS; get back to CONNECT mode by typing exit
Q	\Klogoff	Temporarily quit logging the remote session
R	\Klogon	Resume logging the remote session
S	\Kstatus	Show the status of the connection
Ctrl-]	*none*	(or whatever you have set the escape character to be) Type the escape character twice to send it once to the host

MS-DOS Kermit Keyboard Verbs

Table I-4 shows the assignments of Kermit verbs to keys for use during terminal emulation. To change the assignments, use the SET KEY command, for example:

```
MS-Kermit>set key \324 \Kexit
```

This assigns the EXIT function to the F10 key (see Table I-9 for keyboard scan codes).

Table I-4 MS-DOS Kermit Keyboard Verbs

Verb	*Meaning*	*Assignment*
`\Kbreak`	Send a BREAK signal	*`Alt-B`*, *`Ctrl-Break`*
`\KdecDo`	DEC Do key	*none*
`\KdecF6`	DEC F6 key	*none*
`\KdecF7`	DEC F7 key (etc., up to 14)	*none*
`\KdecF17`	DEC F17 key (etc., up to 20)	*none*
`\KdecFind`	DEC Find key	*none*
`\KdecHelp`	DEC Help key	*none*
`\KdecInsert`	DEC Insert key	*none*
`\KdecNext`	DEC Next Screen key	*none*
`\KdecPrev`	DEC Prev Screen key	*none*
`\KdecRemove`	DEC Remove key	*none*
`\KdecSelect`	DEC Select key	*none*
`\Kdnarr`	Transmit what DEC Down-Arrow key sends	*`Down-Arrow`*
`\Kdnone`	Roll screen down one line	*`Ctrl-Page-Down`*
`\Kdnscn`	Roll down (forward) to next screen	*`Page Down`*
`\Kdos`	"Push" to DOS	*`Ctrl-]P`*
`\Kdump`	Text or graphic screen dump*	*`Ctrl-End`*
`\Kendscn`	Roll down to end of screen memory	*`End`*
`\Kexit`	Escape back from CONNECT mode	*`Alt-X`*
`\Kgold`	Transmit what DEC Gold key sends	*`F1`*
`\Khangup`	Drop DTR so modem will hang up phone	*none*
`\Khelp`	Display CONNECT help message	*`Alt-H`*
`\Kholdscrn`	Toggle hold screen mode	*none*

Table I-4 MS-DOS Kermit Keyboard Verbs (continued)

Verb	*Meaning*	*Assignment*
`\Khomscn`	Roll up to top of screen memory	*Home*
`\Kkp0`	DEC keypad 0	*Shift-F7*
`\Kkp1`	DEC keypad 1	*Shift-F3*
`\Kkp2`	DEC keypad 2	*Shift-F4*
`\Kkp3`	DEC keypad 3	*Shift-F5*
`\Kkp4`	DEC keypad 4	*F9*
`\Kkp5`	DEC keypad 5	*F10*
`\Kkp6`	DEC keypad 6	*Shift-F1*
`\Kkp7`	DEC keypad 7	*F5*
`\Kkp8`	DEC keypad 8	*F6*
`\Kkp9`	DEC keypad 9	*F7*
`\Kkpcoma`	DEC keypad comma	*Shift-F2*
`\Kkpdot`	DEC keypad dot (period)	*Shift-F8*
`\Kkpenter`	DEC keypad Enter	*Shift-F6*
`\Kkpminus`	DEC keypad minus	*F8*
`\Klbreak`	Send a "long BREAK" signal	*Ctrl-]L*
`\Klfarr`	Transmit what DEC Left-Arrow key sends	*Left-Arrow*
`\Klogoff`	Turn off session logging	*Ctrl-]Q*
`\Klogon`	Turn on session logging	*Ctrl-]R*
`\Kmodeline`	Toggle modeline off/on	*Keypad minus*
`\Knethold`	Put a network connection on hold	*Alt-n*
`\Knull`	Send a null (ASCII 0)	*Ctrl-]0*
`\Kpf1`	PF1, same as Gold	*F1*
`\Kpf2`	DEC PF2 key	*F2*
`\Kpf3`	DEC PF3 key	*F3*
`\Kpf4`	DEC PF4 key	*F4*
`\Kprtscn`	Toggle screen printing	*Ctrl-Print-Screen*
`\Kreset`	Reset terminal emulator to initial state	*Alt-=*
`\Krtarr`	Transmit what DEC Right-Arrow key sends	*Right-Arrow*
`\Kstatus`	Display STATUS message	*Alt-S*

Table I-4 MS-DOS Kermit Keyboard Verbs (continued)

Verb	*Meaning*	*Assignment*
`\Kterminalr`	Invoke user-defined macro TERMINALR	*none*
`\Kterminals`	Invoke user-defined macro TERMINALS	*none*
`\Ktermtype`	Toggle terminal type	*Alt-minus*
`\Kuparr`	Transmit what DEC Up-Arrow key sends	*Up-Arrow*
`\Kupone`	Roll screen up one line	*Ctrl-Page-Up*
`\Kupscn`	Roll up (back) to previous screen	*Page Up*

The `\Kdump` verb operates differently for text and graphics screens. Text screens are appended to `KERMIT.SCN` (or whatever other filename you have given in your most recent SET DUMP command), and separated from previous material by a formfeed (Ctrl-L). Graphics screens are saved in TIFF 5.0 format to files named `TEKPLT01.TIF`, `TEKPLT02.TIF`, and so on. Note that `\Kdump` cannot be invoked for graphics screens by typing *Ctrl-]F*. Use Ctrl-End for that.

ASCII Character Codes

Key: *Dec* = decimal value, *Hex* = hexadecimal value, ^X = Ctrl-X.

Table I-5 ASCII Character Codes, ANSI X3.4-1986

Dec	*Hex*	*Name*	*Char*	*Dec*	*Hex*	*Char*	*Dec*	*Hex*	*Char*	*Dec*	*Hex*	*Char*
000	00	NUL	^@	032	20	SP	064	40	@	096	60	‘
001	01	SOH	^A	033	21	!	065	41	A	097	61	a
002	02	STX	^B	034	22	"	066	42	B	098	62	b
003	03	ETX	^C	035	23	#	067	43	C	099	63	c
004	04	EOT	^D	036	24	$	068	44	D	100	64	d
005	05	ENQ	^E	037	25	%	069	45	E	101	65	e
006	06	ACK	^F	038	26	&	070	46	F	102	66	f
007	07	BEL	^G	039	27	’	071	47	G	103	67	g
008	08	BS	^H	040	28	(	072	48	H	104	68	h
009	09	HT	^I	041	29	)	073	49	I	105	69	i
010	0A	LF	^J	042	2A	*	074	4A	J	106	6A	j
011	0B	VT	^K	043	2B	+	075	4B	K	107	6B	k
012	0C	FF	^L	044	2C	,	076	4C	L	108	6C	l
013	0D	CR	^M	045	2D	-	077	4D	M	109	6D	m
014	0E	SO	^N	046	2E	.	078	4E	N	110	6E	n
015	0F	SI	^O	047	2F	/	079	4F	O	111	6F	o
016	10	DLE	^P	048	30	0	080	50	P	112	70	p
017	11	DC1	^Q	049	31	1	081	51	Q	113	71	q
018	12	DC2	^R	050	32	2	082	52	R	114	72	r
019	13	DC3	^S	051	33	3	083	53	S	115	73	s
020	14	DC4	^T	052	34	4	084	54	T	116	74	t
021	15	NAK	^U	053	35	5	085	55	U	117	75	u
022	16	SYN	^V	054	36	6	086	56	V	118	76	v
023	17	ETB	^W	055	37	7	087	57	W	119	77	w
024	18	CAN	^X	056	38	8	088	58	X	120	78	x
025	19	EM	^Y	057	39	9	089	59	Y	121	79	y
026	1A	SUB	^Z	058	3A	:	090	5A	Z	122	7A	z
027	1B	ESC	^[	059	3B	;	091	5B	[	123	7B	{
028	1C	FS	^\	060	3C	<	092	5C	\	124	7C	|
029	1D	GS	^]	061	3D	=	093	5D	]	125	7D	}
030	1E	RS	^^	062	3E	>	094	5E	^	126	7E	~
031	1F	US	^_	063	3F	?	095	5F	_	127	7F	DEL

National Replacement Character Sets

Table I-6 shows the National Replacement Character (NRC) sets available to MS-DOS Kermit users as terminal character sets. These 7-bit sets are identical to ASCII (Table I-5) except in the positions shown in this table. ASCII is United States ANSI X3.4-1986. British, French, German, Italian, Norwegian, Portuguese, Spanish, and Swedish are ISO 646 national versions registered in the ISO International Register of Coded Character Sets. The others are taken from DEC VT terminal manuals and other sources.

Table I-6 National Replacement Character Sets

Row/Column	2/03	4/00	5/11	5/12	5/13	5/14	5/15	6/00	7/11	7/12	7/13	7/14
Decimal	35	64	91	92	93	94	95	96	123	124	125	126
Hexadecimal	23	40	5B	5C	5D	5E	5F	60	7B	7C	7D	7E
ASCII	#	@	[	\	]	^	_	‘	{	|	}	~
British	£	@	[	\	]	^	_	‘	{	|	}	~
Dutch	£	¾	ÿ	½	|	^	_	‘	¨	*f*	¼	’
Finnish	#	@	Ä	Ö	Å	Ü	_	é	ä	ö	å	ü
French	£	à	°	ç	§	^	_	µ	é	ù	è	¨
Fr-Canadian	#	à	â	ç	ê	î	_	ô	é	ù	è	û
German	#	§	Ä	Ö	Ü	^	_	‘	ä	ö	ü	ß
Italian	£	§	°	ç	é	^	_	ù	à	ò	è	ì
Norwegian	§	@	Æ	Ø	Å	^	_	‘	æ	ø	å	|
Portuguese	#	’	Ã	Ç	Õ	^	_	‘	ã	ç	õ	~
Spanish	£	§	¡	Ñ	¿	^	_	‘	°	ñ	ç	~
Swedish	#	É	Ä	Ö	Å	Ü	_	é	ä	ö	å	ü
Swiss	ù	à	é	ç	ê	î	è	ô	ä	ö	ü	û

IBM PC and PS/2 Code Pages and ISO Latin Alphabet 1

Table I-7 shows the IBM PC and PS/2 code pages, listed in alphabetical order for easy reference. To enter these characters, look up the desired character in your current code page, hold down the Alt key and type the three-digit *decimal* number (listed in the *Dec* column) on the PC's numeric keypad, or use the PC's "dead key" combinations listed in the back of the DOS manual (DOS 3.30 or later) with your KEYB driver. Only the special alphabetic and punctuation characters are listed. For ASCII characters in the range 0–127, see Table I-5. CP437 is the original IBM PC code page. CP850 is the multilingual code page. CP860 is for Portugal. CP863 is for French Canada. CP865 is for Norway. The Latin-1 column shows ISO 8859-1 Latin Alphabet 1, used by MS-DOS Kermit as a file transfer character set. *If a code appears in the Latin-1 column, MS-DOS Kermit can translate the corresponding character.*

Table I-7 IBM PC and PS/2 Code Pages

		CP437	**CP850**	**CP860**	**CP863**	**CP865**	**Latin-1**
Char	*Name*	*Dec Hex*	*Dec Hex*	*Dec Hex*	*Dec Hex*	*Dec Hex*	*Dec Hex*
á	a-acute	160 A0	160 A0	160 A0		160 A0	225 E1
Á	A-acute		181 B5	134 86			193 C1
â	a-circumflex	131 83	131 83	131 83	131 83	131 83	226 E2
Â	A-circumflex		182 B6	143 8F	132 84		194 C2
æ	ae digraph	145 91	145 91			145 91	230 E6
Æ	AE digraph	146 92	146 92			146 92	198 C6
à	a-grave	133 85	133 85	133 85	133 85	133 85	224 E0
À	A-grave		183 B7	145 91	142 8E		192 C0
å	a-ring	134 86	134 86			134 86	229 E5
Å	A-ring	143 8F	143 8F			143 8F	197 C5
ã	a-tilde		198 C6	132 84			227 E3
Ã	A-tilde		199 C7	142 8E			195 C3
ä	a-diaeresis	132 84	132 84			132 84	228 E4
Ä	A-diaeresis	142 8E	142 8E			142 8E	196 C4
ç	c-cedilla	135 87	135 87	135 87	135 87	135 87	231 E7
Ç	C-cedilla	128 80	128 80	128 80	128 80	128 80	199 C7
é	e-acute	130 82	130 82	130 82	130 82	130 82	233 E9

Table I-7 IBM PC and PS/2 Code Pages (continued)

Char	*Name*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
É	E-acute	144 90	144 90	144 90	144 90	144 90	201 C9
ê	e-circumflex	136 88	136 88	136 88	136 88	136 88	234 EA
Ê	E-circumflex		210 D2	137 89	146 92		202 CA
è	e-grave	138 8A	138 8A	138 8A	138 8A	138 8A	232 E8
È	E-grave		212 D4	146 92	145 91		200 C8
ë	e-diaeresis	137 89	137 89		137 89	137 89	235 EB
Ë	E-diaeresis		211 D3		148 94		203 CB
í	i-acute	161 A1	161 A1	161 A1		161 A1	237 ED
Í	I-acute		214 D6	139 8B			205 CD
î	i-circumflex	140 8C	140 8C		140 8C	140 8C	238 EE
Î	I-circumflex		215 D7		168 A8		206 CE
ı	i-dotless		213 D5				
ì	i-grave	141 8D	141 8D	141 8D		141 8D	236 EC
Ì	I-grave		222 DE	152 98[20]			204 CC
ï	i-diaeresis	139 8B	139 8B		139 8B	139 8B	239 EF
Ï	I-diaeresis		216 D8		149 95		207 CF
ñ	n-tilde	164 A4	164 A4	164 A4		164 A4	241 F1
Ñ	N-tilde	165 A5	165 A5	165 A5		165 A5	209 D1
ó	o-acute	162 A2	162 A2	162 A2	162 A2	162 A2	243 F3
Ó	O-acute		224 E0	159 9F			211 D3
ô	o-circumflex	147 93	147 93	147 93	147 93	147 93	244 F4
Ô	O-circumflex		226 E2	140 8C	153 99		212 D4
ò	o-grave	149 95	149 95	149 95		149 95	242 F2
Ò	O-grave		227 E3	169 A9			210 D2
ø	o-slash		155 9B			155 9B	248 F8
Ø	O-slash		157 9D			157 9D	216 D8
õ	o-tilde		228 E4	148 94			245 F5
Õ	O-tilde		229 E5	153 99			213 D5

[20]CP860 I-grave is also erroneously listed as 139/8B in the IBM DOS 3.30 manual.

Table I-7 IBM PC and PS/2 Code Pages (continued)

Char	*Name*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
ö	o-diaeresis	148 94	148 94			148 94	246 F6
Ö	O-diaeresis	153 99	153 99			153 99	214 D6
ú	u-acute	163 A3	163 A3	163 A3	163 A3[21]	163 A3[21]	250 FA
Ú	U-acute		233 E9	150 96			218 DA
û	u-circumflex	150 96	150 96		150 96	150 96	251 FB
Û	U-circumflex		234 EA		158 9E		219 DB
ù	u-grave	151 97	151 97	151 97	151 97[22]	151 97[22]	249 F9
Ù	U-grave		235 EB	157 9D	157 9D		217 D9
ü	u-diaeresis	129 81	129 81	129 81	129 81	129 81	252 FC
Ü	U-diaeresis	154 9A	154 9A	154 9A	154 9A	154 9A	220 DC
ý	y-acute		236 EC				253 FD
Ý	Y-acute		237 ED				221 DD
ÿ	y-diaeresis	152 98	152 98			152 98	255 FF
ß	German ss	225 E1[23]	225 E1	225 E1[23]	225 E1[23]	225 E1[23]	223 DF
α	Greek alpha	224 E0		224 E0	224 E0	224 E0	
β	Greek beta	225 E1	225 E1[24]	225 E1	225 E1	225 E1	
δ	Greek delta	235 EB		235 EB	235 EB	235 EB	
ε	Greek epsilon	238 EE		238 EE	238 EE	238 EE	
ϕ	Greek fi	237 ED		237 ED	237 ED	237 ED	
Φ	Greek Fi	232 E8		232 E8	232 E8	232 E8	
Γ	Greek Gamma	226 E2		226 E2	226 E2	226 E2	
μ	Greek mu	230 E6	230 E6	230 E6	230 E6	230 E6	
π	Greek pi	227 E3		227 E3	227 E3	227 E3	
σ	Greek sigma	229 E5		229 E5	229 E5	229 E5	

[21]CP863 and 865 also erroneously list u-acute as 151/97.

[22]CP863 and 865 erroneously list u-acute in this position.

[23]Use Greek beta.

[24]Use German double s.

Table I-7 IBM PC and PS/2 Code Pages (continued)

Char	*Name*	**CP437** *Dec Hex*	**CP850** *Dec Hex*	**CP860** *Dec Hex*	**CP863** *Dec Hex*	**CP865** *Dec Hex*	**Latin-1** *Dec Hex*
Σ	Greek Sigma	228 E4		228 E4	228 E4	228 E4	
τ	Greek tau	231 E7		231 E7	231 E7	231 E7	
Θ	Greek Theta	233 E9		233 E9	233 E9	233 E9	
Ω	Greek Omega	234 EA		234 EA	234 EA	234 EA	
ð	Icelandic eth		208 D0				240 F0
Ð	Icelandic Eth		209 D1				208 D0
þ	Icelandic thorn		231 E7				254 FE
Þ	Icelandic Thorn		232 E8				222 DE
¡	!-inverted	173 AD	173 AD	173 AD		173 AD	161 A1
¿	?-inverted	168 A8	168 A8	168 A8		168 A8	191 BF
«	Left guillemot	174 AE	174 AE	174 AE	174 AE	174 AE	171 AB
»	Right guillemot	175 AF	175 AF	175 AF	175 AF		187 BB
´	Acute accent		239 EF		161 A1		180 B4
¸	Cedilla				165 A5		184 B8
¨	Diaeresis		249 F9		164 A4		168 A8
¯	Macron		238 EE		167 A7		175 AF
§	Paragraph sign	021 15	021 15	021 15	021 15	021 15	167 A7
¶	Pilcrow sign	020 14	020 14	020 14	020 14	020 14	182 B6
ª	Fem. ordinal	166 A6	166 A6	166 A6		166 A6	170 AA
º	Masc. ordinal	167 A7	167 A7	167 A7		167 A7	186 BA
©	Copyright		184 B8				169 A9
®	Trade mark		169 A9				174 AE
£	English Pound	156 9C	156 9C	156 9C	156 9C	156 9C	163 A3
¥	Japanese Yen	157 9D	190 BE				165 A5
¢	Cent sign	155 9B	189 BD	155 9B	155 9B		162 A2
¤	Currency sign		207 CF		152 98		164 A4
Pt	Peseta sign	158 9E		158 9E		158 9E	
ƒ	Florin sign	159 9F	159 9F		159 9F	159 9F	

Cyrillic Character Sets Used by MS-DOS Kermit

Table I-8 shows the characters of the ISO 8859-5 Latin/Cyrillic Alphabet (which is supported as a transfer character set by MS-DOS Kermit); Microsoft Code Page 866 (which is supported as a file character set by MS-DOS Kermit); "Old KOI-8", a commonly used mainframe character set; and the "Short KOI" equivalents that are used for displaying Russian words on ASCII devices, in which Cyrillic letters are shown as lowercase ASCII and Roman letters as uppercase ASCII, for example "`Protokol Peredahi Fajlov` Kermit" is written "protokol pereda~i fajlow KERMIT".

The character names are taken from ISO Standard 8859-5, modified to show upper- or lowercase typographically rather than spelling out UPPER CASE and LOWER CASE for each letter. Unfortunately, the same name is used by ISO for two different characters: Cyrillic I (1) looks like a backwards Roman letter "N" and Cyrillic I (2) looks like a Roman letter I.

CP866 and the KOI character sets lack the Macedonian and Serbocroation letters, as well as the old Cyrillic letters and one of the Ukrainian letters that are found in ISO 8859-5. The CP866 characters B0 through DF (hex) are identical to the line- and box-drawing characters of IBM CP437. The 8-bit Latin/Cyrillic and CP866 character sets all include ASCII as their first 128 characters (except $ is replaced by ¤ in KOI-8). *If a code appears in the ISO column, Kermit can translate the corresponding character.*

Table I-8 Cyrillic Character Sets

Character	*Name*	**ISO** *Dec Hex*	**CP866** *Dec Hex*	**KOI-8** *Dec Hex*	**Short KOI**
`A`	Cyrillic A	176 B0	128 80	225 E1	a
`a`	Cyrillic a	208 D0	160 A0	193 C1	a
`B`	Cyrillic Be	177 B1	129 81	226 E2	b
`b`	Cyrillic be	209 D1	161 A1	194 C2	b
`V`	Cyrillic Ve	178 B2	130 82	247 F7	w
`v`	Cyrillic ve	210 D2	162 A2	215 D7	w
`G`	Cyrillic Ghe	179 B3	131 83	231 E7	g
`g`	Cyrillic ghe	211 D3	163 A3	199 C7	g
`D`	Cyrillic De	180 B4	132 84	228 E4	d
`d`	Cyrillic de	212 D4	164 A4	196 C4	d

Table I-8 Cyrillic Character Sets (continued)

Character	*Name*	**ISO** *Dec Hex*	**CP866** *Dec Hex*	**KOI-8** *Dec Hex*	**Short KOI**
E	Cyrillic Ie	181 B5	133 85	229 E5	e
e	Cyrillic ie	213 D5	165 A5	197 C5	e
fl	Cyrillic Io	161 A1	240 F0		
¿	Cyrillic io	241 F1	241 F1		
+	Cyrillic Zhe	182 B6	134 86	246 F6	v
=	Cyrillic zhe	214 D6	166 A6	214 D6	v
Z	Cyrillic Ze	183 B7	136 87	250 FA	z
z	Cyrillic ze	215 D7	167 A7	218 DA	z
I	Cyrillic I (1)	184 B8	136 88	233 E9	i
i	Cyrillic i (1)	216 D8	168 A8	201 C9	i
J	Cyrillic Short I	185 B9	137 89	234 EA	j
j	Cyrillic Short i	217 D9	169 A9	202 CA	j
K	Cyrillic Ka	186 BA	138 8A	235 EB	k
k	Cyrillic ka	218 DA	170 AA	203 CB	k
L	Cyrillic El	187 BB	139 8B	236 EC	l
l	Cyrillic el	219 DB	171 AB	204 CC	l
M	Cyrillic Em	188 BC	140 8C	237 ED	m
m	Cyrillic em	220 DC	172 AC	205 CD	m
N	Cyrillic En	189 BD	141 8D	238 EE	n
n	Cyrillic en	221 DD	173 AD	206 CE	n
O	Cyrillic O	190 BE	142 8E	239 EF	o
o	Cyrillic o	222 DE	174 AE	207 CF	o
P	Cyrillic Pe	191 BF	143 8F	240 F0	p
p	Cyrillic pe	223 DF	175 AF	208 D0	p
R	Cyrillic Er	192 C0	144 90	242 F2	r
r	Cyrillic er	224 E0	224 E0	210 D2	r
S	Cyrillic Es	193 C1	145 91	243 F3	s
s	Cyrillic es	225 E1	225 E1	211 D3	s
T	Cyrillic Te	194 C2	146 92	244 F4	t

Table I-8 Cyrillic Character Sets (continued)

Character	*Name*	**ISO** *Dec Hex*	**CP866** *Dec Hex*	**KOI-8** *Dec Hex*	**Short KOI**
t	Cyrillic te	226 E2	226 E2	212 D4	t
U	Cyrillic U	195 C3	147 93	245 F5	u
u	Cyrillic u	227 E3	226 E3	213 D5	y
F	Cyrillic Ef	196 C4	148 94	230 E6	f
f	Cyrillic ef	228 E4	228 D4	198 C6	f
X	Cyrillic Ha	197 C5	149 95	232 E8	h
x	Cyrillic ha	229 E5	229 E5	200 C8	h
C	Cyrillic Tse	198 C6	150 96	227 E3	c
c	Cyrillic tse	230 E6	230 E6	195 C3	c
H	Cyrillic Che	199 C7	151 97	254 FE	~
h	Cyrillic che	231 E7	231 E7	222 DE	~
W	Cyrillic Sha	200 C8	152 98	251 FB	{
w	Cyrillic sha	232 E8	232 E8	219 DB	{
&	Cyrillic Shcha	201 C9	153 99	253 FD	}
7	Cyrillic shcha	233 E9	233 E9	221 DD	}
~	Cyrillic Hard Sign	202 CA	154 9A		
`	Cyrillic hard sign	234 EA	234 EA	207 CF	
Y	Cyrillic Yeri	203 CB	155 9B	249 F9	y
y	Cyrillic yeri	235 EB	234 EB	217 D9	y
{	Cyrillic Soft Sign	204 CC	156 9C	248 F8	x
[	Cyrillic soft sign	236 EC	236 EC	216 D8	x
|	Cyrillic E	205 CD	157 9D	252 FC	|
\	Cyrillic e	237 ED	237 ED	220 DC	|
}	Cyrillic Yu	206 CE	158 9E	224 E0	@
]	Cyrillic yu	238 EE	238 EE	192 C0	@
Q	Cyrillic Ya	207 CF	159 9F	241 F1	q
q	Cyrillic ya	239 EF	239 EF	209 D1	q
	Cyrillic Dze	175 AF			
	Cyrillic dze	255 FF			

Table I-8 Cyrillic Character Sets (continued)

Character	*Name*	**ISO** *Dec Hex*	**CP866** *Dec Hex*	**KOI-8** *Dec Hex*	**Short KOI**
!	Cyrillic I (2)	166 A6			
1	Cyrillic i (2)	246 F6			
	Cyrillic Je	168 A8	244 F4		
˘	Cyrillic je	248 F8	245 F5		
æ	Cyrillic Lje	169 A9			
´	Cyrillic lje	249 F9			
	Cyrillic Nje	170 AA			
	Cyrillic nje	250 FA			
^	Bielorussian Short U	174 AE	246 F6		
6	Bielorussian short u	254 FE	247 F7		
	Macedonian Dze	165 A5			
‰	Macedonian dze	245 F5			
	Macedonian Gje	163 A3			
'	Macedonian gje	243 F3			
	Macedonian Kje	172 AC			
ß	Macedonian kje	252 FC			
%	Serbocroatian Dje	162 A2			
¶	Serbocroatian dje	242 F2			
	Serbocroatian Chje	171 AB			
	Serbocroatian chje	251 FB			
#	Ukrainian Ie	164 A4	242 F2		
3	Ukrainian ie	244 F4	243 F3		
$	Ukrainian Yi	167 A7			
4	Ukrainian yi	247 F7			
	No-break space	160 A0	255 FF		
	Number Acronym	240 F0	252 FC		
§	Paragraph sign	253 FD			
	Soft hyphen	173 AD			

MS-DOS Kermit Keyboard Scan Codes

Table I-9 shows the scan codes recognized by MS-DOS Kermit on the IBM PC and PS/2 USA keyboards, for use with Kermit's SET KEY command. Other keyboards or computers may have different codes. Use SHOW KEY to find out what they are. The columns show the scan code for the key when pressed by itself, with Shift, with Ctrl, with Ctrl and Shift (C-S), with Alt, with Shift and Alt (S-A), with Ctrl and Alt (C-A), and with Ctrl, Shift, and Alt (C-S-A) all at the same time. In the *Key* column, kp refers to the numeric keypad, and Gray indicates the group of keys between the main keyboard and the numeric keypad on enhanced keyboards.

Keys marked with (*) used in conjunction with Ctrl and Ctrl-Shift will produce either the scan codes indicated or no scan code at all, depending on the keyboard model and driver. Older keyboards have no F11 and F12 keys.

Table I-9 Kermit Keyboard Scan Codes for the IBM PC and PS/2

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
' "	39	34			2344	2856	3368	3880
, <	44	60			2355	2867	3379	3891
- _	45	95	31	31	2434	2946	3458	3970
. >	46	62			2356	2868	3380	3892
/ ?	47	63			2357	2869	3381	3893
0) (*)	48	41	1464	1976	2433	2945	3457	3969
1 ! (*)	49	33	1465	1977	2424	2936	3448	3960
2 @	50	64	0	1795	2425	2937	3449	3961
3 # (*)	51	35	1467	1979	2426	2938	3450	3962
4 $ (*)	52	36	1468	1980	2427	2939	3451	3963
5 % (*)	53	37	1469	1981	2428	2940	3452	3964
6 ^	54	94	30	30	2429	2941	3453	3965
7 & (*)	55	38	1471	1983	2430	2942	3454	3966
8 * (*)	56	42	1472	1984	2431	2943	3455	3967
9 ((*)	57	40	1473	1985	2432	2944	3456	3968
; :	59	58			2343	2855	3367	3879
= +	61	43			2435	2947	3459	3971

Table I-9 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
[{	91	123	27	27	2330	2842	3354	3866
\ |	92	124	28	28	2347	2859	3371	3883
] }	93	125	29	29	2331	2843	3355	3867
‘ ~	96	126			2345	2857	3369	3881
a A	97	65	1	1	2334	2846	3358	3870
b B	98	66	2	2	2352	2864	3376	3888
c C	99	67	3	3	2350	2862	3374	3886
d D	100	68	4	4	2336	2848	3360	3872
e E	101	69	5	5	2322	2834	3346	3858
f F	102	70	6	6	2337	2849	3361	3873
g G	103	71	7	7	2338	2850	3362	3874
h H	104	72	8	8	2339	2851	3363	3875
i I	105	73	9	9	2327	2839	3351	3863
j J	106	74	10	10	2340	2852	3364	3876
k K	107	75	11	11	2341	2853	3365	3877
l L	108	76	12	12	2342	2854	3366	3878
m M	109	77	13	13	2354	2866	3378	3890
n N	110	78	14	14	2353	2865	3377	3889
o O	111	79	15	15	2328	2840	3352	3864
p P	112	80	16	16	2329	2841	3353	3865
q Q	113	81	17	17	2320	2832	3344	3856
r R	114	82	18	18	2323	2835	3347	3859
s S	115	83	19	19	2335	2847	3359	3871
t T	116	84	20	20	2324	2836	3348	3860
u U	117	85	21	21	2326	2838	3350	3862
v V	118	86	22	22	2351	2863	3375	3887
w W	119	87	23	23	2321	2833	3345	3857
x X	120	88	24	24	2349	2861	3373	3885
y Y	121	89	25	25	2325	2837	3349	3861
z Z	122	90	26	26	2348	2860	3372	3884

Table I-9 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
Backspace	270	782	127	127	2318	2830	3342	3854
Enter	284	796	10	10	2332	2844	3356	3868
Esc	27	27	27	27	2305	2817	3329	3841
F1	315	852	1374	1886	2408	2920	3432	3944
F2	316	853	1375	1887	2409	2921	3433	3945
F3	317	854	1376	1888	2410	2922	3434	3946
F4	318	855	1377	1889	2411	2923	3435	3947
F5	319	856	1378	1890	2412	2924	3436	3948
F6	320	857	1379	1891	2413	2925	3437	3949
F7	321	858	1380	1892	2414	2926	3438	3950
F8	322	859	1381	1893	2415	2927	3439	3951
F9	323	860	1382	1894	2416	2928	3440	3952
F10	324	861	1383	1895	2417	2929	3441	3953
F11	389	903	1417	1929	2443	2955	3467	3979
F12	390	904	1418	1930	2444	2956	3468	3980
Gray Delete	4435	4947	5523	6035	2467	2979		
Gray Down-Arrow	4432	4944	5521	6033	2464	2976	3488	4000
Gray End	4431	4943	5493	6005	2463	2975	3487	3999
Gray Home	4423	4935	5495	6007	2455	2967	3479	3991
Gray Insert	4434	4946	5522	6034	2466	2978	3490	4002
Gray Left-Arrow	4427	4939	5491	6003	2459	2971	3483	3995
Gray Page Down	4433	4945	5494	6006	2465	2977	3489	4001
Gray Page Up	4425	4937	5508	6020	2457	2969	3481	3993
Gray Right-Arrow	4429	4941	5492	6004	2461	2973	3485	3997
Gray Up-Arrow	4424	4936	5517	6029	2456	2968	3480	3992
Num Lock								
Pause/Break			1280	1792				
PrintScn/SysRq			1394	1906			3328	3840
Scroll Lock								
Space	32	32	32	32	32	32	32	32

Table I-9 Kermit Keyboard Scan Codes for the IBM PC and PS/2 (continued)

Key	*Self*	*Shift*	*Ctrl*	*C-S*	*Alt*	*S-A*	*C-A*	*C-S-A*
Tab	271	783	1428	1940	2469	2981	3493	4005
kp *	311	823	1430	1942	2359	2871	3383	3895
kp +	334	846	1424	1936	2382	2894	3406	3918
kp - (minus)	330	842	1422	1934	2378	2890	3402	3914
kp ., Del	339	851	1427	1939				
kp /	4399	4911	1429	1941	2468	2980	3492	4004
kp 0, Ins	338	850	1426	1938	**	**	**	**
kp 1, End	335	847	1397	1909	**	**	**	**
kp 2, Down-Arrow	336	848	1425	1937	**	**	**	**
kp 3, PgDn	337	849	1398	1910	**	**	**	**
kp 4, Left-Arrow	331	843	1395	1907	**	**	**	**
kp 5	332	844	1423	1935	**	**	**	**
kp 6, Right-Arrow	333	845	1396	1908	**	**	**	**
kp 7, Home	327	839	1399	1911	**	**	**	**
kp 8, Up-Arrow	328	840	1421	1933	**	**	**	**
kp 9, PgUp	329	841	1412	1924	**	**	**	**
kp Enter	4365	4877	5386	5898	2470	2982	3494	4006

The keypad digit keys 0–9 produce the codes listed only when Num Lock is off, except for Alt-key combinations which work the same whether Num Lock is on or off.

Alt-key combinations with keypad digits (marked ** above) produce scan codes based on the digits you type, but only after the Alt key is released. You may press one, two, or three keypad digit keys to enter any value from 1 to 255.

CAUTION: Pressing any of the combinations `Ctrl-Alt-Del`, `Ctrl-Alt-Delete`, `Ctrl-Alt-Shift-Del`, or `Ctrl-Alt-Shift-Delete` will reboot your PC.

Appendix II

Escape Sequences

Appendix II lists the codes and sequences sent and received by MS-DOS Kermit's VT, Heath, and Tektronix terminal emulators on IBM PCs, PS/2s, and compatibles. The first section presents the codes sent by the terminal emulator when you press the "special keys" on your keyboard. The second section lists the escape and control sequences recognized by Kermit's VT and Heath terminal emulators, along with Kermit's responses to them. The third section presents the details of Kermit's Tektronix / VT340 graphics terminal emulator.

Terminal Character Set Terminology and Mechanics

A 7-bit character is one whose 8th (high-order) bit is zero (0). An 8-bit character's 8th bit is one (1). A 7-bit character set has 128 7-bit characters. An 8-bit character set has 256 characters divided into two halves: 128 7-bit characters (normally ASCII) and 128 8-bit characters. Each half has control characters and graphic (printing) characters.

The terminal supports a repertoire of character sets, such as those listed when you type SET TERMINAL CHARACTER-SET ? at the Kermit prompt. From this repertoire, six sets are available at once: C0, C1, G0, G1, G2, and G3. C0 contains 32 7-bit control characters; C1 has 32 8-bit control characters. G0–G3 are graphic character sets of either 94 or 96 characters each. Specific graphic sets from the terminal's repertoire can be designated to G0–G3 by escape sequences from the host (Tables II-9 and II-12) or by Kermit's SET TERMINAL CHARACTER-SET command.

The terminal's active character set is composed of C0 and C1 controls, plus GL (graphics left) and GR (graphics right). GL tells which of the G0–G3 sets is used if a 7-bit graphic character arrives; GR tells which set is used if an 8-bit graphic arrives. Kermit's startup VT320 configuration is: ASCII controls designated to C0, ASCII graphics to both G0 and G1, ISO 6429 controls to C1, and the 96 characters of ISO 8859-1 Latin-1 to both G2 and G3. GL refers to G0 and GR refers to G2, exactly like a real VT320, except Kermit uses Latin-1 rather than DEC MCS in G2 and G3.

Eight-bit character sets can be used in the 7-bit communications environment if the host computer supports *shifting*. In the most common application, the Shift-Out (SO) control character is sent before any 8-bit characters, then 8-bit characters are sent with their 8th bit removed, and then Shift-In (SI) is sent to change back to 7-bit characters. SO invokes G1 into GL, SI invokes G0 back to GL. But since MS-DOS Kermit's startup configuration has ASCII in both G0 and G1 (like a real VT320), SO has no apparent effect. To make SO/SI work in the 7-bit environment, give Kermit the command SET TERMINAL CHARACTER-SET LATIN1 G1 to designate Latin-1 to G1 without affecting G0, G2, or G3. Of course, you can also designate any other of Kermit's graphic character sets to G0, G1, G2, or G3, except it is against the rules to designate a 96-character set to G0 (see Table II-11).

Other shifts can be used to invoke characters from any of G0–G3 into GL or GR, either singly (single shifts) or "permanently" (locking shifts). These are listed in the following tables as LS0 (SI), LS1 (SO), LS1R, SS2, LS2, LS2R, SS3, LS3, and LS3R.

Finally, note that SET TERMINAL CHARACTER-SET TRANSPARENT removes the requirement for a C1 control set and allows all 8-bit characters to be treated as graphics for purposes of display and SET TRANSLATE INPUT.

Escape Sequences Sent by PC Special Keys

This Appendix applies to IBM or compatible systems with IBM keyboards. For systems with DEC LK250 keyboards, see also page 297. The IBM PC version of MS-DOS Kermit emulates DEC VT and Heath terminals. Main keypad keys—letters, digits, punctuation marks, and space, as well as the shifted or control versions of these keys—send a single ASCII value, listed in Table I-5. For "national keyboards," they send values for accented or non-Roman characters according to Kermit's current terminal character set.

Figure II-1 shows the VT and Heath terminal keypads and their IBM PC keyboard equivalents. Each keytop shows the DEC (or Heath) label on top and the IBM label underneath, for example PF1 (DEC) and F1 (IBM). *S-Fn* means "Hold down the Shift key and press the F*n* key." For example, *S-F3* means Shift-F3.

Figure II-1 VT and Heath Terminal Keypads

The following tables list the codes sent by the special keys on a DEC or Heath keyboard: the arrow, keypad, editing, and function keys. Some of these keys send more than one character when you press them. The notation used in the tables to show these characters is simply a list of characters, in which a printable character is shown as itself, and a control character is shown using its ASCII name from Table I-5. Spaces are inserted between each character for ease of reading, but spaces are not actually sent unless noted.

DEC VT-200 and -300 series terminals support a second set of control codes, called "C1" controls, having decimal numeric code values from 128 to 159. Since not all data connections are 8-bit transparent, there are also two-character 7-bit equivalents for these 8-bit control codes. Two of these C1 controls, shown in Table II-1, are used in Kermit's key-oriented \K verbs. The command SET TERMINAL CONTROLS { 7, 8 }, or equivalent control sequences from the host (DECSCL, see Table II-12), determine whether 8-bit controls or their 7-bit equivalents are transmitted when these verbs are executed. Seven-bit controls are always used if you have SET PARITY to EVEN, ODD, MARK, or SPACE. In the tables in this and the following sections, CSI and SS3 stand for the appropriate 7-bit or 8-bit form.

Table II-1 VT-320 C1 Control Codes

Char	*Decimal Value*	*7-bit Equivalent*	*Function*
SS3	143	`ESC O`	Single Shift 3
CSI	155	`ESC [`	Control Sequence Introducer

Arrow Keys

Table II-2 shows the codes sent by the arrow keys. The codes depend on whether the arrow keys are in "application mode" or "cursor mode," which is set by control sequences from the host (SM parameter DECCKM, see Table II-14), or by the MS-DOS Kermit command:

SET TERMINAL ARROW-KEYS { APPLICATION, CURSOR }

The Kermit verbs cause the corresponding keys to transmit the codes based on the arrow-key mode. Kermit uses cursor mode unless directed otherwise.

Table II-2 Codes Sent by DEC and Heath Arrow Keys

Key	*Kermit Verb*	*VT10x/VT320 Cursor Mode*	*VT10x/VT320 Application Mode*	*VT52/Heath All Modes*
Up-Arrow	`\Kuparr`	`CSI A`	`SS3 A`	`ESC A`
Down-Arrow	`\Kdnarr`	`CSI B`	`SS3 B`	`ESC B`
Right-Arrow	`\Krtarr`	`CSI C`	`SS3 C`	`ESC C`
Left-Arrow	`\Klfarr`	`CSI D`	`SS3 D`	`ESC D`

DEC Editing Keys

Table II-3 shows the DEC LK201 keyboard editing keys (found on VT200 and later model DEC terminals), the associated Kermit verbs, and the sequences sent by each keys. The DEC editing keys are not assigned to any IBM keys by default. If you want to use them, you must issue SET KEY commands to assign them to IBM keys, for example "SET KEY \4434 \KDECINSERT" to assign the DEC "Insert Here" key to the IBM gray keypad Insert key (for scan codes, see Table I-9).

Table II-3 Codes Sent by DEC Editing Keys

Key	*Kermit Verb*	*VT320 Mode*	*VT100/VT102/ VT52/H19 Mode*
Find	`\KdecFind`	`CSI 1 ~`	*nothing*
Insert Here	`\KdecInsert`	`CSI 2 ~`	*nothing*
Remove	`\KdecRemove`	`CSI 3 ~`	*nothing*
Select	`\KdecSelect`	`CSI 4 ~`	*nothing*
Prev Screen	`\KdecPrev`	`CSI 5 ~`	*nothing*
Next Screen	`\KdecNext`	`CSI 6 ~`	*nothing*

DEC Numeric Keypad

Table II-4 shows the DEC keyboard numeric keypad keys, the associated Kermit keyboard verbs, the default IBM keyboard assignments, and the codes sent by these keys. The default IBM assignments are to PC function keys (like F1) or shifted function keys (like SF1, meaning Shift-F1), *not* to the IBM numeric keypad. The DEC keypad may be in either "numeric mode" or "application mode." The keypad mode can be set by control sequences from the host (SM parameter DECNKM, see Table II-14) or by the MS-DOS Kermit command:

SET TERMINAL KEYPAD { APPLICATION, NUMERIC }

Kermit's initial keypad mode is numeric.

Table II-4 Codes Sent by the DEC Numeric Keypad

DEC Key	*Kermit Verb*	*IBM Key*	*VT320/10x Numeric Mode*	*VT320/10x Application Mode*	*VT52/Heath Numeric Mode*	*VT52/Heath Application Mode*
PF1/HF7/Blue	\Kgold,\Kpf1	F1	SS3 P	SS3 P	ESC P	ESC P
PF2/HF8/Red	\Kpf2	F2	SS3 Q	SS3 Q	ESC Q	ESC Q
PF3/HF9/Grey	\Kpf3	F3	SS3 R	SS3 R	ESC R	ESC R
PF4/HF1	\Kpf4	F4	SS3 S	SS3 S	ESC S	ESC S
0	\Kkp0	SF7	0	SS3 p	0	ESC ? p
1	\Kkp1	SF3	1	SS3 q	1	ESC ? q
2	\Kkp2	SF4	2	SS3 r	2	ESC ? r
3	\Kkp3	SF5	3	SS3 s	3	ESC ? s
4	\Kkp4	F9	4	SS3 t	4	ESC ? t
5	\Kkp5	F10	5	SS3 u	5	ESC ? u
6	\Kkp6	SF1	6	SS3 v	6	ESC ? v
7	\Kkp7	F5	7	SS3 w	7	ESC ? w
8	\Kkp8	F6	8	SS3 x	8	ESC ? x
9	\Kkp9	F7	9	SS3 y	9	ESC ? y
comma (,)	\Kkpcoma	SF2	,	SS3 l	,	ESC ? l
minus (-)	\Kkpminus	F8	-	SS3 m	-	ESC ? m
period (.)	\Kkpdot	SF8	.	SS3 n	.	ESC ? n
Enter	\Kkpenter	SF6	CR,CRLF	SS3 M	CR,CRLF	ESC ? M

DEC Function Keys

Table II-5 shows the codes sent by the DEC function keys and the Kermit keyboard verbs assigned to them. These are not assigned to any IBM keys, but you may make these assignments using SET KEY commands, for example `set key \2413 \KdecF6` to assign DEC function key 6 to IBM Alt-F6.

Table II-5 DEC Function Key Codes

Key	*Kermit Verb*	*VT320 Mode*	*VT10x/VT52/ Heath Mode*
Hold Screen		*nothing*	*nothing*
Print Screen		*nothing*	*nothing*
Set-Up		*nothing*	*nothing*
F4		*nothing*	*nothing*
F5 (Break)		*nothing*	*nothing*
F6	`\KdecF6`	`CSI 17 ~`	*nothing*
F7	`\KdecF7`	`CSI 18 ~`	*nothing*
F8	`\KdecF8`	`CSI 19 ~`	*nothing*
F9	`\KdecF9`	`CSI 20 ~`	*nothing*
F10	`\KdecF10`	`CSI 21 ~`	*nothing*
F11 (ESC)	`\KdecF11`	`CSI 23 ~`	ESC
F12 (BS)	`\KdecF12`	`CSI 24 ~`	BS
F13 (LF)	`\KdecF13`	`CSI 25 ~`	LF
F14	`\KdecF14`	`CSI 26 ~`	*nothing*
Help	`\KdecHelp`	`CSI 28 ~`	*nothing*
Do	`\KdecDo`	`CSI 29 ~`	*nothing*
F17	`\KdecF17`	`CSI 31 ~`	*nothing*
F18	`\KdecF18`	`CSI 32 ~`	*nothing*
F19	`\KdecF19`	`CSI 33 ~`	*nothing*
F20	`\KdecF20`	`CSI 34 ~`	*nothing*

F6–F20 are the DEC ‘‘User Definable Keys’’ (UDKs), which means their output can be redefined by DECUDK control sequences from the host computer (see page 312).

An attempt is made to safely test for the 101/102 key enhanced keyboard and use it if it is present—the result of this test is available in the Kermit variable `\v(keyboard)`. On en-

hanced keyboards, Kermit separates the individual arrow keys from those on the numeric keypad and also separates the asterisk and forward slash keys on the keypad from those on the regular typewriter keyboard. These special enhanced keyboard (gray) keys are reported as scan codes with 4096 added to the base scan code (see Table I-9).

Special-Purpose Keys

Table II-6 lists the IBM PC keys that have special Kermit functions (verbs) assigned to them, such as screen rollback, screen printing, screen dump, sending BREAK, escaping back from CONNECT mode, and so forth. For a complete list of Kermit verbs, see Table I-4. "Alt –" means hold down Alt and type minus on the upper key rank; this switches between Tektronix graphics mode and the currently selected text terminal type, but does not change most operating parameters of the emulator.

Table II-6 Other IBM Keys Operational in CONNECT Mode

IBM Key	*Kermit Verb*	*Action*
Keypad Del		Send ASCII DEL (rubout) code `\127`
Backspace		Send ASCII DEL (rubout) code `\127` (BS is `\8`)
Keypad –	`\Kmodeline`	Toggle mode line on/off (only if the mode line is enabled and not used by the host)
Alt –	`\Ktermtype`	Switch between text and graphics screens
Alt =	`\Kreset`	Clear screen and reset terminal emulator to starting (setup) state
Alt B	`\Kbreak`	Send a BREAK signal
Alt H	`\Khelp`	Show drop-down help menu
Alt n	`\Knethold`	Place Network connections on hold. This means interrupt the normal PC-to-host connection and invoke the external network control program interface for session management. Works with 3Com(BAPI), Novell(NASI), TES, and UB-Net1
Alt S	`\Kstatus`	Show settings
Alt X	`\Kexit`	Exit Connect mode, back to Kermit prompt
Home	`\Khomscn`	Roll screen up (text down) to beginning of storage
End	`\Kendscn`	Roll screen down (text up) to end of storage
PgUp	`\Kupscn`	Roll screen up (back, earlier) one screen
PgDn	`\Kdnscn`	Roll screen down (forward, later) one screen

Table II-6 Other IBM Keys Operational in CONNECT Mode (continued)

IBM Key	*Kermit Verb*	*Action*
Ctrl-PgUp	`\Kupone`	Roll screen up one line
Ctrl-PgDn	`\Kdnone`	Roll screen down one line
Ctrl-PrtSc	`\Kprtscn`	Toggle on/off copying of received text to printer; "PRN" shows on far right of mode line when activated
Shift-PrtSc		Standard DOS Print-Screen; dump screen image to printer
Ctrl-End	`\Kdump`	Dump image of screen to a disk file or device. Default filename is `KERMIT.SCN` in the current directory. Use command SET DUMP *filename* to change the filename. Text screen images are appended to the file, separated by formfeeds. Graphics screens go to files `TEKPLTnn.TIF` (for example `TEKPLT01.TIF`, `TEKPLT02.TIF`)
unassigned	`\Kholdscrn`	DEC style Holdscreen; the same as typing Control-S

DEC LK250 Keyboards

MS-DOS Kermit provides full-featured emulation of the display features of the DEC VT320 terminal. In many instances, applications that use the display features also use the unique keys found on the DEC LK201 keyboard (used on the VT200 and VT300 series). While Kermit provides extensive keyboard remapping features, three fundamental problems remain:

1. There aren't as many keys on a PC's keyboard as there are on a DEC LK201 keyboard. Some applications may require that control, alternate, or shift patterns be defined for less-used, but needed keys.
2. The IBM-compatible BIOS traps certain keystrokes and processes them before Kermit gets to see them. Some keys, like Num Lock, are not redefinable.
3. Users may need to frequently switch between real DEC terminals and the emulated terminal provided by Kermit, resulting in confused fingers.

The simplest solution would be to attach a DEC LK201 keyboard to the PC. However, the physical connection as well as the protocol for saying "a key was pressed" differ from the standard PC, making this a nearly impossible task.

Fortunately, DEC has another keyboard model, the LK250, which was designed to be used with IBM-compatible systems. In its normal mode, it acts like a regular IBM PC/AT keyboard. Under software control, it can be commanded to convert to "DEC mode," giving us the keys we need. However, the codes returned in DEC mode are not known to Kermit, so two external drivers for the DEC LK250 are provided on the program diskette. MSULK2 controls an LK250 attached to an IBM AT-class system, and MSULKV controls an LK250 attached to a DEC VAXmate system. Select the appropriate driver and include it in your `AUTOEXEC.BAT` file. If you receive an error message from the LK250 driver when loading it, it should be self-explanatory. Common problems are trying to use the wrong driver for your machine or trying to use the driver on a non-AT-class PC.

When you wish to use the LK250 with Kermit, issue the following two commands at the Kermit prompt:

```
MS-Kermit>set key lk
MS-Kermit>set key clear
```

This activates the driver, switches keyboard modes (you should see the SPECIAL light on the keyboard go out), and loads the LK250 key mappings. You can put these two commands in your `MSKERMIT.INI` file (before any other SET KEY commands) if you desire.

Once the keys have been remapped, the LK250 performs the actions shown on the LK250 keycaps when in Kermit connect mode. The correct keyboard state is maintained across local program executions, PUSHes to DOS, etc. The only difference is that the LK250's Compose key is mapped to Escape, since Kermit does not support DEC compose sequences.

LK250 Problem Solving

If you have problems with the keyboard, check to see if you have any of the DEC-supplied keyboard mapping programs in place. They are not required and can cause conflicts if loaded. Also check for "hot-key" pop-up applications. The usual warnings apply here as well.

There is a known interaction on some PS/2 systems and some compatibles with the F13 key. If the next keystroke after pressing F13 doesn't produce the desired result, just press that key again and all should be well.

VT100/200/300 Emulation Escape and Control Sequences

The tables in this section show the MS-DOS Kermit terminal emulator's responses to control characters and escape sequences that are received from the host or, when local echoing is in effect, typed at the keyboard. Graphic (printable) characters, including space, when not part of an escape or control sequence, are displayed on the screen in the current position, possibly after translation according to Kermit's terminal character set. Spaces shown between characters in an escape sequence are there for ease of reading. The actual sequences contain no spaces unless indicated (SP). Unknown escape sequences are absorbed and ignored. Unsupported features are marked by an asterisk (*).

Control Characters

Table II-7 shows the terminal emulator's response to ASCII 7-bit C0 control characters; control codes not shown are ignored. Answerback (ENQ) is a security risk and is not generally supported.

Table II-7 ASCII C0 Control Characters

Name	*Dec*	*Hex*	*Keyboard*	*Description*
NUL	000	00h	^@	Ignored except during transparent printing
ENQ	005	05h	^E	Send answerback sequence (Honeywell only)
BEL	007	07h	^G	Sound DEC-style beep or flash screen
BS	008	08h	^H	Backspace, move cursor left one character
HT	009	09h	^I, Tab	Horizontal tab, move cursor to next tabstop
LF	010	0ah	^J	Linefeed, move cursor down one line
VT	011	0bh	^K	Vertical Tab, treated as a linefeed
FF	012	0ch	^L	Formfeed, treated as a linefeed
CR	013	0dh	^M, Enter	Carriage return, move cursor to column 1
SO / LS1	014	0eh	^N	Locking shift 1, invoke character set in G1 to GL
SI / LS0	015	0fh	^O	Locking shift 0, invoke character set in G0 to GL
DC1	017	11h	^Q	XON flow control, resume communication
DC3	019	13h	^S	XOFF flow control, suspend communication
CAN	024	18h	^X	Cancel escape or control sequence in progress
SUB	026	1ah	^Z	Treated the same as CAN
ESC	027	1bh	^[, Esc	Start escape sequence, cancel any others
DEL	127	7fh	Del	Ignored except during transparent printing

Table II-8 shows the VT320 emulator's response to DEC 8-bit (C1) control codes (ISO 6429) and their equivalent 2-character escape sequences in the 7-bit environment. MS-DOS Kermit recognizes 8-bit controls only if you have SET TERMINAL VT320, SET PARITY NONE, *and* SET TERMINAL DISPLAY 8. Eight-bit controls received under any other conditions are truncated to 7 bits, perhaps with unexpected results. Eight-bit controls not shown in Table II-8 are ignored.

Beware of the action of the OSC, PM, APC, and certain DCS commands. Nothing shows on the screen until an ST arrives, another escape or control sequence begins, or the terminal emulator is manually reset (Kermit verb `\Kreset`, normally assigned to Alt-=). Line noise can occasionally generate these characters or destroy the ST, making it appear that Kermit has stopped working. The same thing happens on a real VT320 terminal.

Table II-8 DEC C1 Control Characters

Name	*8-bit Dec*	*8-bit Hex*	*7-Bit Sequence*	*Description*
IND	132	84h	`ESC D`	Index, move cursor down one line, scrolls
NEL	133	85h	`ESC E`	Next line, like CR/LF, scrolls
HTS	136	88h	`ESC H`	Set horizontal tab at present cursor column
RI	141	8dh	`ESC M`	Reverse index, move cursor up one line, scrolls
SS2	142	8eh	`ESC N`	Single Shift 2, invoke G2 to GL, next character only
SS3	143	8fh	`ESC O`	Single Shift 3, invoke G3 to GL, next character only
DCS	144	90h	`ESC P`	Device Control String introducer
CSI	155	9bh	`ESC [`	Control Sequence Introducer
ST	156	9ch	`ESC \`	String Terminator
OSC	157	9dh	`ESC ]`	Operating System Command, consume through ST
PM	158	9eh	`ESC ^`	Privacy Message, consume through ST
APC	159	9fh	`ESC _`	Applications Program Command, consume through ST

ANSI Escape Sequences

Table II-9 lists the escape sequences recognized in ANSI mode, i.e., when emulating a VT100-series or later DEC terminal. The mnemonic is the DEC name for the escape sequence. Sequences that do not have mnemonics are Kermit extensions.

Table II-9 ANSI Escape Sequences

Escape Sequence	*Mnemonic*	*Description of Action*
`ESC ^L`		Begin Tektronix emulation, clear graphics screen (ignored if SET TERMINAL TEK DISABLE has been given)
`ESC 1`		Same as `ESC ^L`
`ESC 7`	DECSC	Save cursor position, attributes, GL and GR character sets, wrap flag, origin mode (DECOM), SS2 / SS3 shifts
`ESC 8`	DECRC	Restore cursor and other information from DECSC
`ESC # 3`	DECDHL	Double-height and -width line, top half (simulated)
`ESC # 4`	DECDHL	Double-height and -width line, bottom half (simulated)
`ESC # 5`	DECSWL	Single-height and -width line
`ESC # 6`	DECDWL	Double-width single-height line (simulated)
`ESC # 8`	DECALN	Screen alignment test, fill screen with E's
`ESC (` *ident*	SCS	Designates 94-byte character set *ident* to G0. See Table II-11 for character set identifiers
`ESC )` *ident*	SCS	Designates 94-byte character set *ident* to G1
`ESC *` *ident*	SCS	Designates 94-byte character set *ident* to G2
`ESC +` *ident*	SCS	Designates 94-byte character set *ident* to G3
`ESC -` *ident*	SCS	Designates 96-byte character set *ident* to G1
`ESC .` *ident*	SCS	Designates 96-byte character set *ident* to G2
`ESC /` *ident*	SCS	Designates 96-byte character set *ident* to G3
`ESC <`		Exit VT52 mode to previous ANSI mode
`ESC =`	DECKPAM	Enter numeric keypad application mode
`ESC >`	DECKNPNM	Enter numeric keypad numeric mode
`ESC D`	IND	Index. Cursor down one line, can scroll
`ESC E`	NEL	New Line. Cursor to start of line below, can scroll
`ESC SP F`	S7C1T	(ESC, space, F) Don't use 8-bit controls, send 7-bit sequences

Table II-9 ANSI Escape Sequences (continued)

Escape Sequence	*Mnemonic*	*Description of Action*
ESC SP G	S8C1T	(ESC, space, G) Enables output of 8-bit control codes
ESC H	HTS	Set one horizontal tab at current position
ESC M	RI	Reverse Index, cursor up one line, can scroll
ESC N	SS2	Single Shift 2, invoke G2 to GL for next character only
ESC O	SS3	Single Shift 3, invoke G3 to GL for next character only
ESC P	DCS	Start Device Control String command
ESC Z	DECID	Identify terminal. Responses listed in Table II-10
ESC [	CSI	Control Sequence Introducer. See Table II-12
ESC \	ST	String Terminator for device control strings (DCS)
ESC]	OSC	Operating System Command, ignored through ST
ESC ^	PM	Privacy Message, ignored through ST
ESC _	APC	Applications Program Command, ignored through ST
ESC c	RIS	Reset terminal to initial state, hard reset
ESC n	LS2	Locking Shift 2, invoke character set in G2 to GL
ESC o	LS3	Locking Shift 3, invoke character set in G3 to GL
ESC y		Send Honeywell terminal ID (Honeywell only)
ESC |	LS3R	Locking Shift 3 Right, invoke character set in G3 to GR
ESC }	LS2R	Locking Shift 2 Right, invoke character set in G2 to GR
ESC ~	LS1R	Locking Shift 1 Right, invoke character set in G1 to GR

Table II-10 lists the identification strings sent in response to the DECID Identify Terminal query and also to the ANSI primary device attributes request, DA.

Table II-11 shows *idents* used with the SCS (select character set) sequence. DEC VT300-series terminals give the choice of ISO Latin-1 (96) or DEC Supplemental Graphics (94) as the user-preferred supplemental character set (UPSS) via the terminal's Setup menu or host command. MS-DOS Kermit's default UPSS is the Supplemental Graphics set. You can change it with the SET TERMINAL UPSS command.

If the size of the character set does not match the *ident*, nothing happens. Startup defaults are ASCII in G0 and G1, ISO Latin-1 in G2 and G3; GL points to G0, GR points to G2. Activating DEC National Replacement Characters loads the NRC set selected by SET TER-

Table II-10 Terminal Device Reports

Terminal	*Response*	*Description*
VT320	`CSI ? 63;1;2;4;8;9;15 c`	Features: 132 columns, printer, user-defined keys, National Replacement Character sets, sixel graphics, DEC technical characters
VT220	`CSI ? 62;1;2;4;8;9;15 c`	Same features as VT320
VT102	`CSI ? 6 c`	VT102 identifier
VT100	`CSI ? 1 c`	VT100 identifier (Advanced Video)
VT52	`ESC / Z`	VT52 identifier
Heath-19	`ESC / K`	Heath-19 identifier
Tektronix	`CSI ? 63;1;2;4;8;9;15 c`	Same as VT320

MINAL CHARACTER-SET *country* into G0, G1, G2, *and* G3 unless one or more Gn's are specified on the end of the command. Single and Locking shifts can be used to temporarily invoke Gn's into GL or GR.

Table II-11 Character Set Identifiers

Ident	*Size*	*Character Set*
`%5`	94	DEC Supplemental Graphics
`%6`	94	Portuguese NRC
`'`	94	(Or E or 6) Norwegian / Danish NRC
`0`	94	DEC Special Graphics (line drawing)
`1`	94 / 96	Kermit, Alternate-ROM
`2`	94	DEC Special Graphics (line drawing)
`4`	94	Dutch NRC
`5`	94	(Or C) Finnish NRC
`6`	94	(Or E or ') Norwegian / Danish NRC
`7`	94	(Or H) Swedish NRC
`9`	94	(Or Q) French Canadian NRC
`=`	94	Swiss NRC
`>`	94	VT340 DEC Technical set

Table II-11 Character Set Identifiers (continued)

Ident	*Size*	*Character Set*
A	94	British NRC
A	96	The 96 graphic characters of ISO 8859-1 (default in G2, G3)
B	94	ASCII (default in G0, G1)
C	94	(Or 5) Finnish NRC
E	94	(Or ’ or 6) Norwegian / Danish NRC
H	94	(Or 7) Swedish NRC
K	94	German NRC
Q	94	(Or 9) French Canadian NRC
R	94	French NRC
Y	94	Italian NRC
Z	94	Spanish NRC

Control Sequences

Table II-12 shows the control sequences recognized by MS-DOS Kermit’s ANSI terminal emulators. In the 8-bit communications environment, these may be introduced by either the 8-bit CSI character or by the two characters ESC and [(left bracket). In the 7-bit environment, they must always be introduced by ESC [.

Table II-12 ANSI Control Sequences

Control Sequence	*Mnemonic*	*Description of Action*
CSI Pn @	ICH	Insert Pn spaces at and after cursor
CSI Pn A	CUU	Cursor up Pn lines, does not scroll
CSI Pn B	CUD	Cursor down Pn lines, does not scroll
CSI Pn C	CUF	Cursor forward, stays on same line
CSI Pn D	CUB	Cursor backward, stays on same line
CSI Pn E	CNL	ANSI next-line (same as CRLF), do Pn times
CSI Pn F	CPL	Previous-line (reverse index), do Pn times
CSI Pc G	CHA	ANSI cursor to absolute column Pc
CSI Pr; Pc H	CUP	Set cursor to row Pr, column Pc (same as HVP)

Table II-12 ANSI Control Sequences (continued)

Control Sequence	*Mnemonic*	*Description of Action*
`CSI 25; Pc H`		Set cursor to row 25, column Pc. Disables Kermit's mode line. Kermit extension
`CSI Pn I`	CHI	Cursor forward Pn tabstops (Horizontal Index)
`CSI Ps J`	ED	Erase in display, Ps = 0, 1, or 2:
`CSI 0 J`	ED	Erase from cursor to end of screen, inclusive
`CSI 1 J`	ED	Erase from start of screen to cursor, inclusive
`CSI 2 J`	ED	Erase entire screen, reset lines to single width, cursor does not move
`CSI ? Ps J`	DECSED	*Selective erase in display, not supported
`CSI Ps K`	EL	Erase in line:
`CSI 0 K`	EL	Erase from cursor to end of line, inclusive
`CSI 1 K`	EL	Erase from start of line to cursor, inclusive
`CSI 2 K`	EL	Erase entire line, cursor does not move
`CSI ? Ps K`	DECSEL	*Selective erase in line, not supported
`CSI Pn L`	IL	Insert Pn lines preceding current line
`CSI Pn M`	DL	Delete Pn lines from current downward
`CSI Pn P`	DCH	Delete Pn chars from cursor to left
`CSI Pn; Pn R`	CPR	Cursor report (row, column), sent by terminal. Example: home position yields `CSI 1;1 R`
`CSI Pn X`	ECH	Erase Pn chars at and to right of cursor
`CSI Pn a`	CUF	ANSI Cursor Forward Pn columns
`CSI Pn c`	DA	Device Attributes request, see Reports
`CSI > Pn c`	DA	Secondary Device Attributes request, see Reports
`CSI Pr d`	CVA	ANSI Cursor to row Pr, absolute
`CSI Pn e`	CUD	ANSI Cursor down Pn rows
`CSI Pr; Pc f`	HVP	Set cursor to row Pr, column Pc (same as CUP)
`CSI 25; Pc f`		Move cursor to row 25, column Pc. Disables Kermit's mode line. Kermit extension
`CSI Ps g`	TBC	Tabs clear, Ps: 0 = at this position, 3 = all
`CSI 0 i`	MC	Media Copy, print whole screen
`CSI 4 i`	MC	Media Copy, exit transparent print

Table II-12 ANSI Control Sequences (continued)

Control Sequence	*Mnemonic*	*Description of Action*
`CSI 5 i`	MC	Media Copy. Enter transparent print. Sends all characters, except the `CSI 4 i` termination string, to the printer and not the screen. All translation and character set selections are bypassed
`CSI ? Pn i`	MC	Media Copy, DEC printer controls. Lines are printed as they appear on the screen, after character set translation. A screen line is printed when the cursor is moved off it by autowrap, LF, FF, or VT. Pn's are:
`CSI ? 1 i`	MC	Print line containing cursor
`CSI ? 4 i`	MC	Exit auto print (stop echoing to printer)
`CSI ? 5 i`	MC	Enter autoprint (echo screen lines to printer)
`CSI Pa;...Pa h`	SM	Set ANSI mode, see Table II-13
`CSI Pa;...Pa l`	RM	Reset ANSI mode, see Table II-13
`CSI ? Ps;...;Ps h`	SM	Set DEC mode, see Table II-14
`CSI ? Ps;...;Ps l`	RM	Reset DEC mode, see Table II-14
`CSI Ps;...Ps m`	SGR	Select graphic rendition (see Table II-15)
`CSI Ps n`	DSR	Device Status request, see Reports
`CSI ? Ps n`	DECDSR	Device Status request, see Reports
`CSI ! p`	DECSTR	Soft reset of terminal (keeps screen)
`CSI Pa $ p`	DECRQM	Report ANSI mode settings, see Reports
`CSI ? Pd $ p`	DECRQM	Report DEC mode settings, see Reports
`CSI Pl; Pc " p`	DECSCL	Set operating level (set terminal type, soft reset). Pl=61, Pc=0 means VT102. Pl=62 or 63 means VT320, with Pc=0 for 7-bit controls, 1 for 8-bit. If Pc is omitted, Pl=61 means VT100, 62 means VT200, and 63 means VT300
`CSI Ps;...;Ps q`	DECLL	Load LEDs. Ps = 0 clears LEDs 1-4. Ps = 1;2;3;4 sets LEDs 1,2,3,4 on Kermit mode line
`CSI Ps " q`	DECSCA	*Character protection attribute. Not supported
`CSI Pt; Pb r`	DECSTBM	Set top and bottom scrolling margins, respectively. `CSI r` resets margin to full screen
`CSI Ps $ u`	DECRQTSR	Terminal state or color palette request
`CSI & u`	DECRQUPSS	User preferred character set request, see Reports

Table II-12 ANSI Control Sequences (continued)

Control Sequence	*Mnemonic*	*Description of Action*
`CSI Ps $ w`	DECRQPSR	Presentation State request, see Reports
`CSI sol x`	DECREQTPARM	Request Terminal Parameters, see Reports
`CSI 2; Ps y`	DECST	VT102 Confidence Test, exits status line
`CSI 4; Ps;...Ps y`	DECTST	VT320 Confidence Tests, exits status line
`CSI Pn $ |`	DECSCPP	VT340 Columns Per Page. Pn is screen width: 0 or 80 for 80 columns, or else 132. Equivalent to DECCOLM (Table II-14)
`CSI Ps $ }`	DECSASD	Select active status display. Ps = 0 means main display, 1 means status line. Moves cursor to selected display area. Ignored unless the status line has been enabled by `CSI 2 $ ~`
`CSI Pn;...Pn ~`		Invoke Kermit PRODUCT macro, if any, assigning Pn's as macro parameters (Kermit extension). Pn's should be numeric
`CSI Ps $ ~`	DECSSDT	Select status line type:
`CSI 0 $ ~`	DECSSDT	No status line
`CSI 1 $ ~`	DECSSDT	Indicator line (locally owned, Kermit default)
`CSI 2 $ ~`	DECSSDT	Host-writable line

Table II-13 lists the parameters for Set ANSI Mode (`CSI Pa;...Pa h`) and Reset ANSI Mode (`CSI Pa;...Pa l`). Modes with an asterisk (*) are not supported by Kermit.

Table II-13 Set / Reset ANSI Mode Parameters

Parameter	*Mnemonic*	*Mode*	*Set (*`h`*)*	*Reset (*`l`*)*
2	KAM	*Keyboard mode	Locked	Unlocked
3	CRM	Control codes	Act upon	Debug display
4	IRM	Insert / Replace Mode	Insert	Replace
10	HEM	Horizontal editing	n/a	Is always reset
12	SRM	Local echo	Off	On
20	LNM	New line mode	Display CR as CRLF	Display CR as CR

Table II-14 lists the parameters for Set DEC Mode (`CSI ? Pa;...Pa h`) and Reset DEC Mode (`CSI ? Pa;...Pa l`). The parameters 34 (Invoke Kermit Macro) and 38 (Enter / Leave Graphics Terminal Emulation) are Kermit extensions; parameter 38 is ignored if the SET TERMINAL TEK DISABLE command has been given. Automatic switching between 80- and 132-column mode is done directly for supported video adapters; otherwise Kermit runs `COLS80.BAT` or `COLS132.BAT` from the user's path (if they exist), which should contain external commands for changing screen width. Features marked with an asterisk (*) are not supported.

Table II-14 Set / Reset DEC Mode Parameters

Parameter	*Mnemonic*	*Mode*	*Set (*`h`*)*	*Reset (*`l`*)*
1	DECCKM	Cursor keys	Application	Cursor / Numeric
2	DECANM	ANSI mode	VT320 / VT102	VT52
3	DECCOLM	Columns	132 Columns	80 Columns
4	DECSCLM	*Scroll	Smooth	Jump
5	DECSCNM	Screen, whole	Reverse video	Normal
6	DECOM	Origin	Stay in margins	Ignore margins
7	DECAWM	Autowrap	On	Off
8	DECARM	*Autorepeat	On	Off
9	DECINLM	*Interlace	On	Off
18	DECPFF	Printer termination	Form feed	None
19	DECPEX	Printer extent	Screen	Scrolling region
25	DECTCEM	Cursor	Visible	Invisible
34		Invoke Kermit macro:	TERMINALS	TERMINALR
38		Graphics / text	Graphics	Text
42	DECNRCM	NRC Sets	Enable	Disable
66	DECNKM	Numeric keypad	Application	Numeric
68	DECKBUM	*Keyboard	Data processing	Typewriter

Table II-15 lists the parameters for Set Graphic Rendition (`CSI Ps;...Ps m`). Color selection is a Kermit extension. Colors are 1 (red), 2 (green), 4 (blue), or any sum of these, with the result added to 30 for a foreground color or to 40 for a background color, compatible with `ANSI.SYS`. Also see Table 8-4.

Table II-15 Set Graphic Rendition Parameters

Parameter	*Description*
0	Attributes 1, 4, 5, and 7 off, and restore original colors
1	Bold (intensify foreground)
4	Underline (reverse video with non-Mono video adapters)
5	Blink
7	Reverse video, per character
22	Bold off, VT320
24	Underline off, VT320
25	Blinking off, VT320
27	Reverse video off, VT320
30-37	Foreground color = 30 + colors (Kermit extension)
40-47	Background color = 40 + colors (Kermit extension)

Reports

This section lists the control sequences sent to the VT320 to request information and MS-DOS Kermit's responses to these sequences.

`CSI c`

DA, primary device attributes request. Terminal type and features. Responses are the same as to the DECID query, listed in Table II-10.

`CSI > c`

DA, secondary device attributes request. Response is `CSI > 24; 0; 0; 0 c`, meaning VT320, firmware version 0.0.

`CSI 5 n`

DSR, device operating status request. Response is `CSI 0 n`, meaning "no malfunction."

`CSI 6 n`

DSR, device status request, cursor position. Response is `CSI Pr; Pc R`, where `Pr` is the row and Pc is the column. The origin (home) is 1,1.

`CSI ? 6 n`

DECDSR, device status request, cursor position. Same as `CSI 6 n`, but for the VT340 terminal, and Kermit's response is the same.

Table II-16 Keyboard Dialect Report Parameters

Ps	*NRC*	*Ps*	*NRC*	*Ps*	*NRC*	*Ps*	*NRC*
1	US ASCII	14	French	9	Italian	15	Spanish
2	British	4	French Canadian	13	Norwegian / Danish	12	Swedish
8	Dutch	7	German	16	Portuguese	11	Swiss
6	Finnish						

`CSI ? 15 n`

DECDSR, printer status request. Response is `CIS ? 10 n` if the printer is ready, or `CSI ? 11 n` if the printer is not ready.

`CSI ? 25 n`

DECDSR, device status request, User Definable Key (UDK) status. Kermit's response is `CSI ? 20 n`, meaning the UDKs are unlocked (Kermit never locks them).

`CSI ? 26 n`

DECDSR, device status request, keyboard dialect. Response: `CSI ? 27; Ps n`, where `Ps` is an NRC number from Table II-16.

`CSI 2; 2 $ u`

DECRQTSR, VT340 color palette request. Kermit's response is:

```
ESC P 2 $ s p0/p1/...pn ST
```

where `p0`–`pn` are palette colors in the RGB system, three numbers showing the red, green, and blue percentages respectively, e.g., `0;0;0` for black and `20;20;20` for bold black. Dim (regular) hue is 40 and bold hue is 80. This is a long report that might not be acceptable to some communications channels. Kermit always reports in the RGB system. The default color palettes are shown in Table II-17.

`CSI 1 $ u`

DECRQTSR, terminal state request. The host wants the entire state of the VT320. Kermit does not do this; the response is simply `DCS 1 $ ST`.

`DCS Ps $ p data ST`

DECRSTS, restore terminal state. Ignored by MS-DOS Kermit.

`CSI & u`

DECRQUPSS, User Preferred Supplemental Set request. Response is:

```
DCS Ps ! u data ST
```

Ps is 0 for a 94-byte set, or 1 for a 96-byte set. The data is the character set identification string, `A` for ISO Latin-1, `%5` for DEC Supplemental Graphics.

Table II-17 VT340 Color Palette Report Parameters

Palette	*Mono*	*Color*	*Palette*	*Mono*	*Color*
0 (background)	Black	Black	8	Dim gray	Dim gray (bold black)
1	White	Bold blue	9	Gray	Blue
2	White	Bold red	10	Gray	Red
3	White	Bold green	11	Gray	Green
4	White	Bold magenta	12	Gray	Magenta
5	White	Bold cyan	13	Gray	Cyan
6	White	Yellow	14	Gray	Yellow / brown
7 (foreground)	Gray	Gray	15	White	White (bold)

`CSI 1 $ w`

DECRQPSR. Presentation state request at current cursor position. Kermit's response:

`DCS 1 $ u Pr; Pc; Pp; Srend; Satt; Sflag; Pgl; Pgr; Scss; Sdesig ST`

where *Pr* is the present cursor row (counted from origin as 1,1), *Pc* is the cursor column, and *Pp* is the video page, a constant 1 for the VT320, and:

Srend is a hexadecimal number, the sum of: 40h + 8 (if reverse is video on) + 4 (if blinking on) + 2 (if underline on) + 1 (if bold on).

Satt (selective erase, not supported by Kermit) is always reported as 40h.

Sflag is 40h + 8 (if autowrap is pending) + 4 (if SS3 pending) + 2 (if SS2 pending) + 1 (if origin mode is on).

Pgl is the character table currently invoked in GL (0 = G0, 1 = G1, etc.). *Pgr* is the character table invoked in GR.

Scss tells the sizes of the character tables G0–G3: 40h + 8 (if G3 size is 96) + 4 (if G2 is 96) + 2 (if G1 is 96) + 1 (if G0 is 96).

Sdesig is a string of character set identifiers for G0 through G3, with no separators. If NRCs are active the identifiers are as listed in Table II-11.

`CSI 2 $ w`

DECRQPSR, tab stop request. The response is `DCS 2 $ u Pc;Pc;...Pc ST`. The Pc's are column numbers (from 1) where tabs are set. *Note:* the separator "`/`" is sent by a real VT320, but should have been a semicolon.

Table II-18 Request ANSI Mode Responses

Pa	*Mnemonic*	*Meaning*	*Ps*	*State*
2	KAM	Keyboard action (if locked)	0	Unknown mode
3	CRM	Control representation (no debug)	1	Set
4	IRM	Insert/replace mode (if insert mode)	2	Reset
10	HEM	Horizontal editing (perm reset)	3	Permanently set
12	SRM	Send/receive (local echo on)	4	Permanently reset
20	LNM	Newline (if newline on)		

`DCS Ps $ t string ST`

DECRSPS, restore presentation state. Ps is 1 for cursor information, in the form of the `DCS 1 $ w` report above, 2 for tab stop information, in the form of the `DCS 2 $ w` report above. Kermit restores the indicated information; an error may leave the emulator in an inconsistent state. There is no response.

`CSI Pa $ p`

DECRQM, request ANSI modes. Kermit's response is `CSI Pa; Ps $ y`, DECRPM, where the appropriate Pa and Ps are current mode and state, shown in Table II-18.

`CSI ? Pd $ p`

DECRQM, Request DEC modes. Kermit's response is `CSI Pd; Ps $ y`, where `Pd` is a DEC mode from Table II-19 and `Ps` is a state value as in Table II-18.

Table II-19 Request DEC Mode Responses

Pd	*Mnemonic*	*Meaning*	*Pd*	*Mnemonic*	*Meaning*
1	DECCKM	Cursor key mode	18	DECPFF	Print with form feed
2	DECANM	ANSI mode	19	DECPEX	Print extent
3	DECCOLM	132 column mode	25	DECTCEM	Text cursor enabled
4	DECSCLM	Smooth scroll	42	DECNRCM	NRC set
5	DECSCNM	Reverse video	66	DECNKM	Numeric keypad mode
6	DECOM	Origin mode	67	DECBKM	Backarrow sends BS
7	DECAWM	Autowrap	68	DECKBUM	Keyboard usage
8	DECARM	Autorepeat keyboard			

`DCS $ q string ST`

Control function setting request. The response is `DCS Ps $ r string ST`, where `Ps` is 0 for a valid request, 1 for an invalid one. The request string is one of: `$}` (select active status display); `"q` (set character attribute); `"p` (set conformance level); `$~` (set status line type); `r` (set top and bottom margins); or `m` (set graphic rendition).

The string part of the reponse is the same as an incoming command for the corresponding function, but with the leading CSI omitted. The host gets a text string to be repeated back later with a CSI prefix to restore the current state of the selected function. Example: request `DCS $ q r ST` "what are top/bottom margin settings?" Response is: `DCS 0 $ r 1; 24 r ST`, matching the command `CSI 1;24 r`.

`DCS Ps ! u string ST`

Assign User Preferred Supplemental Set. Parameters are the same as for the `CSI & u` request. No response.

`DCS Pc; Pl | Ky1/St1...Kyn/Stn ST`

DECUDK, Set User Definable Keys. Pc = 0 means clear all UDK definitions before starting, Pc = 1 means clear one key. Pl = 0 means lock the keys, Pl = 1 means do not lock; Kermit never locks keys. Kyn / Stn are key identifier and definition string. *Kyn* is two ASCII digits, 17-34, the DEC key number for DEC F6 through DEC F20; F6 = 17, etc., as in Table II-5. *Stn* is the definition written as a pair of hexadecimal digits per definition byte; upper- or lowercase A–F are the same. Example:

```
DCS 1; 1 | 28/48656c70 ST
```

defines DEC Help key (Kermit verb `\KdecHelp`) as the four characters "Help" without erasing other UDK definitions. The lock indicator, Pl, is ignored by Kermit. The maximum length for a definition string is 60 hex characters. Kermit sends no response.

`CSI sol x`

DECREQTPARM, request terminal parameters, VT102 only. Kermit's response (DECREPTPARM) is:

```
CSI sol; par; nbits; xspeed; rspeed; clkmul; flags x
```

where *sol* is 1 (terminal reports sent on request), or 2 (this is a report). *par* tells the parity setting: 1 (none), 2 (space), 3 (mark), 4 (odd), 5 (even). *nbits* is 1 for 8-bit characters, 2 for 7-bit. The transmit and receive speeds xspeed and rspeed are given by a code: 0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, and 128 corresponding to speeds of 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, 19200, and 38400 baud or above. *clkmul* is always 1. *flags* are always reported as 0.

`DCS Ps; ...Ps { string ST`

Download or clear soft characters. No response, not supported by Kermit.

DEC VT52 Emulator Escape Sequences

Table II-20 shows the valid escape sequences for VT52 mode, which is entered with the command SET TERMINAL TYPE VT52, or upon receipt of `CSI ? 2 l` (`l` is lowercase L) when the terminal emulator is in VT100/200/300 mode.

Table II-20 VT52 Escape Sequences

Escape Sequence	*Description of Action*
`ESC 7`	Save cursor position
`ESC 8`	Restore cursor position
`ESC A`	Cursor up
`ESC B`	Cursor down
`ESC C`	Cursor right
`ESC D`	Cursor left
`ESC F`	Enter graphics mode
`ESC G`	Exit graphics mode
`ESC H`	Cursor home
`ESC I`	Reverse linefeed
`ESC J`	Erase to end of screen
`ESC K`	Erase to end of line
`ESC V`	Print cursor line
`ESC X`	Exit printer controller (transparent print)
`ESC Y` *row column*	Direct cursor addressing, offset from Space
`ESC W`	Enter printer controller (transparent print)
`ESC Z`	Identify (response is `ESC / Z`)
`ESC ^`	Enter autoprint mode (printer echoes screen)
`ESC _`	Exit autoprint mode
`ESC ]`	Print Screen
`ESC =`	Enter alternate keypad mode
`ESC >`	Exit alternate keypad mode
`ESC <`	Return to ANSI mode

Heath/Zenith-19 Emulator Escape Sequences

The Heath/Zenith-19 terminal combines VT52 and ANSI features. Enter Heath emulation with the Kermit command SET TERMINAL TYPE HEATH-19. Escape sequences for non-ANSI mode (similar to the DEC VT52) are listed in Table II-21, and for ANSI mode (similar to the DEC VT100) in Table II-23. Heath ANSI mode includes line and character insertion and deletion for rapid screen updates, similar to VT102 and higher DEC terminals, and includes the ability to let the host change the cursor type, a feature not found on VT terminals. Items marked by an asterisk (*) are not supported.

Table II-21 Heath-19 Functions While in Non-ANSI Mode

Escape Sequence	*Mnemonic*	*Description of Action*
`ESC A`	HCUU	Cursor up
`ESC B`	HCUD	Cursor down
`ESC C`	HCUF	Cursor forward, stays on same line
`ESC D`	HCUB	Cursor backward, stays on same line
`ESC E`	HCD	Clear display
`ESC F`	HEGM	Enter graphics mode
`ESC G`	HXGM	Exit graphics mode
`ESC H`	HCUH	Cursor home
`ESC I`	HRI	Reverse index
`ESC J`	HEOP	Erase to end of page
`ESC K`	HEOL	Erase to end of line
`ESC L`	HIL	Insert line
`ESC M`	HDL	Delete line
`ESC N`	HDCH	Delete character
`ESC O`	HERM	Exit insert character mode
`ESC Y` *row column*	HDCA	Direct cursor addressing, offset from Space
`ESC Z`	HID	Identify terminal (response is `ESC / K`)
`ESC b`	HBD	Erase from beginning of display
`ESC j`	HSCP	Save cursor position
`ESC k`	HRCP	Set cursor to saved position
`ESC l`	HEL	Erase entire line

Table II-21 Heath-19 Functions While in Non-ANSI Mode (continued)

Escape Sequence	*Mnemonic*	*Description of Action*
`ESC n`	HCPR	Cursor position report request
`ESC o`	HEBL	Erase beginning of line
`ESC p`	HERV	Enter reverse video mode
`ESC q`	HXRV	Exit reverse video mode
`ESC r Bn`	HMBR	*Modify baud rate, not supported
`ESC t`	HEKS	*Enter keypad shifted mode, not supported
`ESC u`	HXKS	*Exit keypad shifted mode, not supported
`ESC v`	HEWA	Wrap around at end of line
`ESC w`	HXWA	Discard at end of line
`ESC x Ps`	HSM	Set Mode. See Table II-22
`ESC y Ps`	HRM	Reset Mode. See Table II-22
`ESC z`	HRAM	Reset to power-up configuration
`ESC =`	HAKM	Enter alternate keypad mode
`ESC >`	HXAM	Exit alternate keypad mode
`ESC <`	EAM	Enter ANSI mode. See Table II-23
`ESC @`	HEIM	Enter insert character mode
`ESC [`	EHS	*Enter hold screen mode, not supported
`ESC \`	HXHS	*Exit hold screen mode, not supported
`ESC {`	EK	*Keyboard enable, not supported
`ESC }`	HDK	*Keyboard disable, not supported
`ESC ]`	HX25	*Transmit 25th line, not supported
`ESC #`	HXMP	*Transmit page, not supported

Table II-22 lists the Heath / Zenith-19 Set / Reset Mode command parameters. The *Ps* column shows the final character from the Set / Reset Mode command, `ESC x Ps` (Set) or `ESC y Ps` (Reset).

Table II-22 Heath Set / Reset Mode Parameters

Ps	*Mnemonic*	*Mode*	*Set (x)*	*Reset (y)*
1	HSM / HRM	25th Line	Enable	Disable & clear
2		*Keyclick	Off	On
3		*Holdscreen	Enabled	Disabled
4		Cursor type	Block	Underline
5		Cursor on / off	On	Off
6		*Keypad	Shifted	Unshifted
7		Alt Appl Keypad	Enabled	Disabled
8		Linefeed	Display LF as CRLF	Display LF as LF
9		Newline Mode	Display CR as CRLF	Display CR as CR

Table II-23 shows Heath-19 functions while in ANSI mode. The Heath-19 terminal emulator enters ANSI mode when it receives the EAM escape sequence, `ESC <`, and leaves ANSI mode when it receives PEHM, `ESC [ ? 2 h`. Kermit's Heath-19 emulator is in ANSI mode by default. Non-ANSI sequences are executed even while in ANSI mode, but ANSI sequences are merely printed on the screen in non-ANSI mode.

Table II-23 Heath-19 Functions While in ANSI Mode

Escape Sequence	*Mnemonic*	*Description of Action*
`ESC [ Pn A`	CUP	Cursor up Pn lines
`ESC [ Pn B`	CUD	Cursor down Pn lines
`ESC [ Pn C`	CUF	Cursor forward Pn columns
`ESC [ Pn D`	CUB	Cursor backward Pn columns
`ESC [ Pr; Pc H`	CUP	Cursor to absolute row, column
`ESC [ Ps J`	ED	Erase in Display, see DEC description, Table II-12
`ESC [ Ps K`	EL	Erase in Line, see DEC description, Table II-12
`ESC [ Pn L`	IL	Insert Pn lines at and below current line
`ESC [ Pn M`	DL	Delete Pn lines at and below current line
`ESC [ Pn P`	DCH	Delete Pn chars at and to right of cursor
`ESC [ Pr; Pc f`	HVP	Cursor to absolute row, column
`ESC [ Ps h`	SM	Set mode. See Table II-22
`ESC [ Ps l`	RM	Reset mode. See Table II-22
`ESC [ Ps m`	SGR	Set graphics rendition: 0 = exit reverse video, 7 = enter reverse video, 10 = enter special graphics, 11 = exit special graphics.
`ESC [ 6 n`	CPR	Cursor report request, returns `ESC [ Pr; Pc R`
`ESC [ p`	PXMT	*Transmit page, not supported
`ESC [ q`	PX25	*Transmit 25th line, not supported
`ESC [ Ps r`	PMBR	*Modify baud rate, not supported
`ESC [ s`	PSCP	Save cursor position and attributes
`ESC [ u`	PRCP	Restore cursor position and attributes
`ESC [ z`	PRAM	Reset to power-up configuration
`ESC [ ? 2 h`	PEHM	Revert to Heath-19 non-ANSI mode
`ESC [ > Ps h`	SM	Same as `ESC x Ps` in non-ANSI mode
`ESC [ > Ps l`	RM	Same as `ESC y Ps` in non-ANSI mode

Tektronix Graphics Escape Sequences

The MS-DOS Kermit Tektronix terminal emulator, available for the IBM PC, PS/2, and compatibles with most common types of graphics adapters, has the characteristics of real Tektronix 4010 and 4014 terminals plus extensive additions from the DEC VT340 and Human Data Systems 2000/3000 series of terminals. VT340 Sixel graphics are supported, but the ReGIS graphics command set is not.

Video adapters are used in EGA, CGA, or text mode, whichever is the highest mode supported by the adapter. Text characters received during graphics mode are truncated to 7 bits and displayed in ASCII; international characters are not supported. Kermit commands related to graphics are described in Chapters 8 and 17.

The VT320 main text terminal emulator switches automatically to graphics terminal emulation whenever certain escape sequences are received from the host (or typed at the keyboard in local-echo mode) and returns automatically to your text terminal type if certain other escape sequences are received in graphics mode. These sequences are shown in Table II-24. When graphics mode is terminated by escape sequence, your previous text terminal emulation is restored automatically and, if your video board had sufficient memory to save it, your previous text screen is restored too. You can disable Kermit's automatic text/graphics mode switching with the command SET TERMINAL TEK DISABLE.

Table II-24 Host-Initiated Tektronix Mode Entry and Exit

Sequence	*Description*
`ESC ^L`	Enter (Tektronix screen clear command)
`ESC 1`	Enter (Same as `ESC ^L`)
`ESC [ ? 38 h`	Enter (VT340 command to enter Tektronix mode)
`DCS Pn q`	Enter (Start of VT340 Sixel command string, `Pn` is a digit)
`ESC [ ? 38 l`	Exit (VT340 command to exit Tektronix mode)
`ESC ^X`	Exit (Enter bypass mode)
`ESC 2`	Exit Tektronix mode, return to text mode

Control Characters

Table II-25 shows the Tektronix emulator's responses to 7-bit (C0) and 8-bit (C1) control characters. Eight-bit controls are converted internally to their 7-bit equivalents, as in Table II-8. Control characters not listed are ignored.

Table II-25 ASCII C0 Control Characters

Name	*Dec*	*Hex*	*Keyboard or 7-bit Equivalent*	*Description*
NUL	000	00h	^@	Ignored
BEL	007	07h	^G	Sound DEC style beep
BS	008	08h	^H	Backspace, move cursor left one character, 8 dots, can be destructive
HT	009	09h	^I	Treated as a single space
LF	010	0ah	^J	Linefeed, move cursor down one line, 8 dots
VT	011	0bh	^K	Vertical Tab, treated as a line feed
FF	012	0ch	^L	Formfeed, erase screen, home cursor
CR	013	0dh	^M	Carriage return, move cursor to column 1
DC1	017	11h	^Q	XON flow control, resume communication
DC3	019	13h	^S	XOFF flow control, suspend communication
CAN	024	18h	^X	Return to text terminal, only if in Tek sub-mode. Ignored if regular Tek terminal (SET TERMINAL TEK)
SUB	026	1ah	^Z	Treated as a CAN
ESC	027	1bh	^[	Escape, start escape sequence, cancel any others
FS	028	1ch	^\	Enter point-plotting mode
GS	029	1dh	^]	Enter line-drawing mode
RS	030	1eh	^^	Enter incremental line-drawing mode
US	031	1fh	^_	Enter Tek text mode (leave line/point drawing). There is no character-set translation, and only ASCII characters are recognized
DCS	144	90h	ESC P	Device Control String introducer
CSI	155	9bh	ESC [	Control Sequence Introducer
ST	156	9ch	ESC \	String Terminator

Table II-26 The Tektronix Coordinate System

HIY	*LOY*	*HIX*	*LSBXY*	*Coordinates Sent*					
Same	Same	Same	Same					LOX	
Same	Same	Same	Diff		LSBXY,	LOY,		LOX	(4014)
Same	Same	Diff	Same			LOY,	HIX,	LOX	
Same	Same	Diff	Diff		LSBXY,	LOY,	HIX,	LOX	(4014)
Same	Diff	Same	Same			LOY,		LOX	
Same	Diff	Same	Diff		LSBXY,	LOY,		LOX	(4014)
Same	Diff	Diff	Same			LOY,	HIX,	LOX	
Same	Diff	Diff	Diff		LSBXY,	LOY,	HIX,	LOX	(4014)
Diff	Same	Same	Same	HIY,				LOX	
Diff	Same	Same	Diff	HIY,	LSBXY,	LOY,		LOX	(4014)
Diff	Same	Diff	Same	HIY,		LOY,	HIX,	LOX	
Diff	Same	Diff	Diff	HIY,	LSBXY,	LOY,	HIX,	LOX	(4014)
Diff	Diff	Same	Same	HIY,		LOY,		LOX	
Diff	Diff	Same	Diff	HIY,	LSBXY,	LOY,		LOX	(4014)
Diff	Diff	Diff	Same	HIY,		LOY,	HIX,	LOX	
Diff	Diff	Diff	Diff	HIY,	LSBXY,	LOY,	HIX,	LOX	(4014)

The Tektronix Coordinate System

The plot commands are characters that specify the position to move the beam to in terms of horizontal (X) and vertical (Y) coordinates. All moves except the one immediately after the GS character are with a visible trace.

For 4010-like devices, the positions are from 0 to 1023 for both X and Y, although only 0 to 780 are visible for Y due to screen geometry. The screen is 10.23 by 7.80 inches, and coordinates are sent as 1 to 4 characters. The position, 0-1023, is broken up into two 5-bit quantities: HIX and LOX for an X coordinate, HIY and LOY for the Y coordinate. These 5-bit quantities have values in the range 0-32, corresponding to ASCII control characters, so each of these values has a number added to it to make it printable and to distinguish the different coordinates from one another: HIY is increased by 32, LOY by 96, HIX by 32, and LOX by 64.

For 4014-like devices, the positions are from 0 to 4095. Each movement is a multiple of 4 positions unless an extra quantity called LSBXY is included, which contains the least

significant two bits of the 12-bit X and Y coordinates, encoded as: 96 + (LSBY x 4) + LSBX. This makes the 4014 compatible with the 4010 in that a full-sized plot fills the screen.

Coordinates are constructed differently, depending on whether any of their elements changed since last time, as shown in Table II-26.

LOY must be sent if HIX has changed so that the TEKTRONIX knows the HIX byte (in the range of 20h-3fh) is HIX and not HIY. LOY must also be sent if LSBXY has changed, so that the 4010 will ignore LSBXY and accept LOY.

Plotting Commands

The Tektronix commands for drawing lines or dots are introduced by single control characters, as is the command for entering Tektronix text mode:

^] (GS) *coordinates*
Enter Tek line plot mode. The first move is with the beam off (“move to”) and subsequent coordinates are reached with the beam on (“draw to”). Terminate drawing upon reception of CR, LF, RS, US, FS, or CAN.

^^ (RS) { SP, P } *letters*
Enter Tek line incremental plot mode. SP (a space character) after RS means move without drawing; a capital P after the RS means draw. The letters after the space or P are interpreted as shown in Table II-27. Exit drawing upon reception of CR, LF, RS, US, FS, or CAN. Example: RS SP J J J means move three Tek positions left and down (three southwest steps) with the pen up (move invisibly).

^\ (FS) *coordinates*
Draw a dot at each coordinate. Point plotting mode. Like GS but does not join end points with lines. Exit drawing upon reception of CR, LF, RS, US, FS, or CAN.

^_ (US)
Exit Tek line plot mode and return to Tek text mode.

Table II-27 Incremental Plot Mode Commands

Char	*Action*	*Char*	*Action*
D	Up (north)	E	Right and up (NE)
H	Down (south)	F	Left and up (NW)
B	Left (west)	I	Right and down (SE)
A	Right (east)	J	Left and down (SW)

Graphics Escape Sequences

Table II-28 lists the escape sequences recognized by Kermit while in graphics mode.

Table II-28 Graphics Escape Sequences

Escape Sequence	*Description of Action*
ESC ^E	Request Tek status report. Report is 20h X Y 0dh for non-text mode; 24h X Y 0dh for text mode. X Y is Tek-style cursor position
ESC ^L	Enter Tektronix sub-mode, clear Tek screen
ESC ^X	Turn on bypass mode (no screen characters). Exit Tek submode if entered via ESC ^L
ESC ^Z	Turn on GIN crosshairs
ESC ?	Substitute for DEL, for 7-bit systems
ESC P	Device Control Sequence introducer (DCS). See section on sixel graphics
ESC Z	Report terminal type. Response is same as for VT320. See Table II-10
ESC @ – ESC M	(@,A–M) Select rectangular fill pattern 1–14. See ESC /..y, Table II-29
ESC \	String Terminator (ST, of DCS items)
ESC '	Select line type: 11111111 11111111
ESC a	10101010 10101010
ESC b	11110000 11110000
ESC c	11111010 11111010
ESC d	11111111 11001100
ESC e	11111100 10010010
ESC x	User-defined (by ESC / Pn a)
ESC y	User-defined (by ESC / Pn b)
ESC z	User-defined (by ESC / Pn c)

Graphics Control Sequences

Table II-29 shows the MS-DOS Kermit's graphics-related control sequences. Tektronix and VT340 control sequences begin with CSI or ESC [. Kermit also accepts HDS 2000/3000 sequences that begin with ESC /.

Certain text-related control sequences are also accepted in graphics mode. These work by assuming the graphics screen has the same row and column dimensions as the text ter-

minal emulator's screen. Since the two screens generally do not share common divisors, some misregistration can occur. Graphics-mode characters use an 8x8 dot bit-mapped pattern. Only ASCII characters are available. The text-mode control sequences supported by Kermit include ICH, CUU, CUD, DUF, DUB, CNL, CPL, CHA, CUP, ED, EL, DCH, ECH, CUF, CVA, CUD, and HVP (see Table II-12).

Table II-29 Graphics Control Sequences

Control Sequence	*Description of Action*
`ESC / P1; P2;...; P8 C`	Define user fill pattern. Use low 8 bits of each Pn. P1 is top of fill, plotted least significant bit first. The pattern is 8x8 dots. Omitted Pn's are 0's
`ESC / P1; P2;...; P8 D`	Define second user fill pattern, as above
`ESC / Pn a`	Set the user definable line drawing pattern to be selected by `ESC x`. Pn is the decimal ASCII representation of a 16-bit number, the bit pattern to use for drawing lines (1 = dot, 0 = space). The pattern is used cyclically, least significant bit first
`ESC / Pn b`	Set user definable line pattern for `ESC y`
`ESC / Pn c`	Set user definable line pattern for `ESC z`
`ESC / Pd d`	Set pixel operation code. 0 means draw 1's in foreground color, skip 0's. 1 means draw 1's in background color, skip 0's. 2 means XOR 1's with foreground color, skip 0's. 3 means write absolute, 1's in foreground and 0's in background color
`ESC / Pn h`	Set. Pn = 2 means destructive space, 9 means destructive backspace. Both are off (reset) by default. Both are turned on if the command SET TERMINAL GRAPHICS CHARACTER-WRITING OPAQUE is given or if Tek mode is entered via sixel DCS while in text mode
`ESC / Pn l`	Reset, Pn's as in Set
`ESC / Px; Py; Ph; Pw x`	Draw rectangle. Px,Py is lower left corner, Ph is height, Pw is width. Line is drawn as foreground dots only
`ESC / Px; Py; Ph; Pw; Pp y`	Fill a rectangle. Px,Py is lower left corner, Ph is height, Pw is width, Pp is fill pattern number (1–14), all in Tek coordinates. Fill patterns are shown in Table II-30. All are 8x8 tiling pixel patterns locked to the top left of the screen and are drawn using the current pixel operation code (`ESC / Pd d`)

Table II-29 Graphics Control Sequences (continued)

Control Sequence	*Description of Action*
`ESC / Px; Py; Ph; Pw; Pp z`	Fill a rectangle, as for `ESC / ...y` and then add a line border as for `ESC / ...x`
`CSI Pn; Pn m`	Set screen colors. Values accepted are 0, 1, 30-37, and 40-47, as in Table II-15. This command sets palette 0 to the background color and palette 7 to the foreground color
`CSI 2; 2 $ u`	Request VT340 color palette. Report is as for VT340. Defaults are as in Table II-17
`CSI ? 34 h`	Invoke Kermit macro TERMINALS, if defined. Exits CONNECT mode
`CSI ? 34 l`	Invoke macro TERMINALR, as above
`CSI ? 38 l`	Exit Tek mode to text terminal emulator, only if Tek mode was invoked from text emulator by `ESC [ ? 38 h`, by `ESC 1`, or by a Sixel DCS
`CSI ? 256 n`	Request screen size report, MS-DOS Kermit only. Report is `ESC [ ? 256; Ph; Pw; Pc n` for graphics systems, where `Ph` is screen height in dots, `Pw` is screen width in dots, `Pc` is number of colors (0, 1 or 16, for none, mono, EGA/VGA). Report is `ESC [ ? 24;80;0 n` for pure text monochrome systems

Table II-30 shows the rectangle fill patterns for Draw Rectangle or Fill Rectangle commands (Table II-29).

Table II-30 Tektronix Rectangle Fill Patterns

`Pp`	*Pattern*	`Pp`	*Pattern*
0	Use default	8	Vertical cross-hatch
1	Solid	9	Checkerboard
2	Gray (50% dots)	10	Dotted, sparse
3	Left to right slant	11	Horizontal herringbone
4	Right to left slant	12	Vertical herringbone
5	Horizontal lines	13	User defined (by `ESC / Pn C`)
6	Vertical lines	14	User defined (by `ESC / Pn D`)
7	Slanted cross-hatch		

Sixel Graphics

In addition to Tektronix graphics, Kermit supports VT340 sixel graphics, so called because six pixels (picture elements, i.e. screen dots) are encoded per character. A sixel graphics command is a device control sequence:

```
DCS P1; P2; P3 q string ST
```

where `DCS` is the 8-bit `DCS` character itself or its 7-bit equivalent, `ESC P`, and `ST` is the 8-bit `ST` character or `ESC \`. In this command, P1 and P3 are ignored. P2 = 0 or 2 means draw 0 bits in background; 1 means skip them.

The *string* is a sixel command string, containing mixtures of:

Sixel characters
: (3fh..7eh, lower 6 bits+3fh, displayed as six dots vertically, least significant bit at the top after subtracting 3fh). `?` is all zeros, `@` is top line only, ~ is all 6 bits on. The initial sixel character is placed at the top left of the current 8×8 text cell; subsequent characters work to the right without wrapping. Writing below the screen bottom results in overwriting the bottom strip.

`! Pn Pc`
: The sixel character Pc is drawn Pn times. Pn is a decimal number like 9, 99, or 1234. For example, "`0;0;0q!640~`" writes a solid block line across the screen (Pn = 640, Pc = "~").

`" Pc; Pad; Ph; Pv`
: Raster attributes (all ignored).

`$` Go to left margin.

`-` (minus) Go to left margin and 6 dots down.

Control characters
: Perform the function, stay in sixel mode. Linefeed increments by 8 dots (text cell size).

Escape sequences
: These are permitted within the string and occur without disruption.

`# Pc; Pu; Px; Py; Pz`
: Set color palette.

In the set color palette sequence, `Pc` is the color palette number, 0–255 (0 is background, 7 is normal foreground); only 0–15 are predefined. Palettes can be selected by the substring `# Pc` followed by a non-numeric character other than a semicolon, where `Pc` is the palette number, from 0 to 255. `Pu` is the color system, 1 = HLS (Hue Lightness Saturation), and 2 = RGB (Red Green Blue).

In the RGB system, Px is the percentage of red, 0-100, Py the percentage of green, and Pz the percentage of blue. If any color exceeds 50%, the bold bit is turned on for the ensemble.

In the HLS system, Px is the hue angle, 0-360 degrees. The colors are mapped around the color wheel in 60 degree segments as hues: 0-29 = blue, 30-89 = magenta (blue + red), 90-149 = red, 150-209 = yellow (red + green), 210-269 = green, 270-329 = cyan (green + blue), 330-359 = blue. Py is lightness, 0-100%, and Pz is Saturation, 0-100%. Table II-31 illustrates the relation between lightness and saturation. Py = Pz = 50 gives the widest spectrum.

Table II-31 Lightness versus Saturation

Lightness	*Saturation 51-100*	*11-50*	*0-10*
86-100	Bold white	Bold white	Bold white
71-85	Bold hue	Bold white	Bold white
57-70	Bold hue	Gray (dim white)	Gray
43-56	Bold hue	Dim hue	Black
29-42	Dim hue	Gray	Gray
14-28	Dim hue	Black	Black
0-13	Black	Black	Black

Sixel character plotting begins at the upper left of the current text cell. Thus, either Tek or ANSI cursor steering commands can be employed to locate the starting position. The ESC [..m coloring sequence can occur within a sixel string and acts on the current fore- and background colors, storing them in palettes 7 and 0. Sixel dots are stored by ORing the palette value with the palette value already existing in that dot, except all black writes black absolutely. At the completion of a sixel DCS, the screen colors are reset to palette 7 and 0 for foreground and background, respectively.

Appendix III

The MS-DOS Kermit Distribution Diskette

The 5.25-inch DSDD 360K MS-DOS Kermit diskette included at the back of this book contains the following files:

`READ.ME`
: Explanation of the files on the diskette, similar to this Appendix. Be sure to read this file (use the DOS TYPE command) in case the diskette has been updated since the publication of this book.

`KERMIT.EXE`
: The MS-DOS Kermit program for the IBM PC family, the IBM PS/1 and PS/2, and compatibles, ready to run.

`MSKERMIT.INI`
: A sample initialization file for MS-DOS Kermit. Includes many of the macro definitions from Chapter 14. Edit this file to suit your needs and preferences.

KERMIT.HLP
: A summary of the commands and functions of MS-DOS Kermit. You may view this file with the DOS or Kermit TYPE command, a text editor or word processing program (in ASCII mode), or print it on your printer.

KERMIT.BWR
: A list of known problems and limitations of the current release of MS-DOS Kermit, and hints for getting around them. If you are having problems using MS-DOS Kermit, read this file. You might find a solution or workaround.

VT300.INI
: An initialization file for setting up your PC keyboard as much like a DEC VT200 or VT300 LK201 keyboard as possible, including assigning the DEC function keys F6-F20 to IBM PC function keys, and also making assignments for the DEC editing keypad, numeric keypad, and arrow keys. If you use Kermit to access host-based applications that require you to type DEC function or editing keys, TAKE this file from the `MS-Kermit>` prompt or put the command `TAKE VT300.INI` in your `MSKERMIT.INI` file. Use this as a model for creating new keyboard setups.

VT300.DOC
: Documentation for `VT300.INI`.

HAYES.SCR
: A TAKE command file to be used for dialing Hayes modems. Invoked by the DIAL macro that is defined in `MSKERMIT.INI`. Should be stored on your current disk and directory or in your DOS PATH.

DIALUPS.TXT
: A sample dialing directory for use with the DIAL command. This file does not contain any real phone numbers. If you want to have a dialing directory, edit this file to make entries for the computers or services that you actually use. The format is: one line per entry, up to four "words" per line. The words are separated by one or more spaces. If you want a word itself to contain spaces, enclose it in curly braces.

  The first word is the host or service name, the second is the phone number (as you would type it to your modem), the third is the speed in bits per second (e.g., 2400), and the fourth is the parity to use: even, odd, mark, none, or space. If you omit the speed and parity, Kermit uses its current settings.

  To use the dialing directory, just type DIAL, a space, and then the name of the host or service you want to call. For further information, see the section on dialing in the `KERMIT.HLP` file.

`COLS132.BAT`
: A DOS Batch file invoked automatically by Kermit if the host sends a "switch to 132-column mode" escape sequence or if you give the `SET TERMINAL WIDTH 132` command to MS-DOS Kermit, but only if Kermit does not already have built-in knowledge of your video adapter. As supplied, this batch only prints a message. You must fill it in with the appropriate DOS commands to put your screen into 132-column mode (as supplied by the manufacturer of your video adapter). The `COLS132.BAT` file must reside on your current disk and directory, or in your DOS PATH.

`COLS80.BAT`
: A DOS Batch file that is invoked automatically if the host sends the escape sequence to change to 80-column mode or you give the command `SET TERMINAL WIDTH 80` to MS-DOS Kermit, but only if Kermit does not have built-in knowledge of your video display adapter. As supplied, this batch file just prints a message. You must fill it in with the appropriate DOS commands to change your screen from 132-column mode to 80-column mode. Otherwise, `COLS80.BAT` works like `COLS132.BAT`.

`MSULK2.HLP`
: Documentation on using the DEC LK250 keyboard and driver.

`MSULK2.COM`
: A driver for the DEC LK250 keyboard attached to IBM PCs and PS/2s.

`MSULKV.COM`
: A driver for the DEC LK250 keyboard attached to a VAXmate.

`CP437.TXT`, `CP850.TXT`, `LATIN1.TXT`
: These three files contain the special characters from IBM Code Page 437, IBM Code Page 850, and ISO 8859 Latin Alphabet 1, with annotations listing the decimal, row/column, octal, and hexadecimal values, and a description of each character so you can compare how it is displayed with what it is supposed to be. Read Chapter 13 to learn about international character sets, and then practice setting and changing your PC code page, TYPEing these files on your PC, transferring them to other computers, displaying these files on the remote computer through Kermit's terminal emulator, transferring them back again, and printing them. This will demonstrate the problems of using international characters in a multivendor computing environment and how to use Kermit to cope with them.

Additional MS-DOS Kermit material is available on diskette or magnetic tape from Columbia University (see page *xxii*): patches, demos, printer utilities, key settings files for various host environments and host-based software packages, and more.

Trademarks

Adobe Systems Incorporated, Mountain View, CA: PostScript.
Aldus Corporation, Seattle, WA: PageMaker.
Apple Computer, Cupertino, CA: Apple, Apple II, Macintosh.
Ashton-Tate Corporation, Torrance, CA: dBase, dBase III, dBase III Plus, dBase IV.
AT&T Information Systems, Morristown, NJ: Bell, UNIX, Touch-Tone, StarLAN, StarGROUP.
Bibliographic Retrieval Services, Latham, NY: BRS.
Cabletron Systems Inc., Rochester, NH: Cabletron, E3010.
Compaq Computer Corporation: COMPAQ.
CompuServe, Inc. (an H&R Block Company), Columbus, OH: CompuServe.
Data General Corporation, Westboro, MA: DG/1.
Datapoint Corporation, San Antonio, TX: ARCNET.
Dell Computer Corporation: Dell, Dell System.
Dialog Information Services, Inc., Palo Alto, CA: DIALOG.
Digital Equipment Corporation, Maynard, MA: The Digital logo, DEC, PDP-11, VAX, VMS, VT52, VT100, VT102, VT220, VT320, Rainbow, VAXmate, DECserver, DECstation, PATHWORKS, ULTRIX.
Digital Equipment Corporation, Intel Corporation, Xerox Corporation: Ethernet.
Dow Jones and Company, Princeton, NJ: Dow-Jones News/Retrieval Service.
FTP Software, Wakefield, MA: PC/TCP, TNGLASS.
Grid: Compass.
Groupe Bull: Honeywell, VIP.
Hayes Microcomputer Products, Inc., Atlanta, GA: Hayes Smartmodem 1200 and 2400.
Heath Company, Benton Harbor, MI: Heath-19.
Henson Associates, Inc., New York, NY: Kermit. The Kermit protocol was named after Kermit the Frog, star of THE MUPPET SHOW television series. The name "Kermit" is used by permission of Henson Associates, Inc., New York.

Hercules Computer Technology: Hercules.
Hewlett-Packard Co., Palo Alto, CA: HP-150, Portable Plus.
Insignia Solutions, Inc.: SoftPC.
Intel Corp., Santa Clara, CA: Intel 8080, 8086, 8088, 80286, 80386, 80486, OpenNET.
Interlan, Boxborough, MA: Interlan
International Business Machines Corp., Armonk, NY: IBM, Series/1, VM/CMS, MVS/TSO, PC-DOS, OS/2, 3270, 3705, 3725, System/370, IBM PC, IBM PC/XT, IBM PC/AT, IBM PC*jr*, IBM RT PC, IBM PS/1, IBM PS/2, IBM 7171, Token Ring, NETBIOS, CGA, MCGA, EGA, VGA, XGA.
Lotus Development Corporation, Cambridge, MA: Lotus 1-2-3.
MCI Communication Corporation, Piscataway, NJ: MCI Mail.
Microcom, Inc., Norwood, MA: Microcom Networking Protocol, MNP.
Microsoft Corporation, Redmond, WA: Microsoft, MS, MS-DOS, XENIX, MASM, Microsoft Mouse, Microsoft Windows, Microsoft Word, LAN Manager.
Nippon Electric Corporation: NEC PC9801
Northgate: Northgate
Novell, Inc., Provo, UT: Novell Network, NetWare 286, NetWare 386, LAN Workplace for DOS, Excelan.
Olivetti: Olivetti.
Oracle Corporation, Belmont, CA: Oracle.
Prime Computer, Natick, MA: PRIME, PRIMOS.
Quarterdeck Office Systems, Santa Monica, CA: DesqView, QEMM.
Sun Microsystems, Inc., Mountain View, CA: Sun Microsystems, Sun Workstation, SUN-4, SUNOS.
Telecom Canada, Ottawa, ON, Canada: Datapac.
Teletype Corporation, Skokie, IL: Teletype.
3Com Corporation, Santa Clara, CA: 3Com, Bridge Applications Programmer Interface (BAPI).
Toshiba: Toshiba T1000.
TYMNET, Inc., San Jose, CA: TYMNET.
Ungermann-Bass, Santa Clara, CA: Net/One.
US Sprint Communications Company Limited Partnership, Shawnee Mission, KS: US Sprint, SprintNet, Telenet.
Ventura Corporation: Ventura Publisher.
WordPerfect Corporation, Orem, UT: WordPerfect.
The Wollongong Group, Palo Alto, CA: WIN/TCP, WIN/ROUTE.
Xerox Corporation, Stamford, CT: Ethernet.
Zenith Data Systems, Glenview, IL: Zenith-19 Terminal.

The text of this book was entered using MS-DOS Kermit on IBM PC/ATs and PS/2s into GNU EMACS on a SUN-4 UNIX system. The text was formatted for PostScript using the Scribe document preparation system. GNU EMACS is a product of the Free Software Foundation, 675 Massachusetts Avenue, Cambridge, MA 02139, USA. MS-DOS Kermit is a product of Kermit Distribution, Columbia University Center for Computing Activities, 612 West 115th Street, New York, NY 10025, USA. Scribe is a commercial product of Scribe Systems, Inc., Suite 240, Commerce Court, Four Station Square, Pittsburgh, PA 15219-1119, USA.

Index

Contents

Illustrations

Tables