BLAST®

Professional UNIX User Manual

BLAST® SOFTWARE

The File Transfer Authority

49 Salisbury Street West
Pittsboro, NC 27312

Manual #2MNUNIX
1/00

BLAST, Inc.
49 Salisbury Street West
P.O. Box 818
Pittsboro, North Carolina 27312

SALES: (800) 242 - 5278
FAX: (919) 542 - 0161
Technical Support: (919) 542 - 3007
E-mail: info@blast.com
World Wide Web: http://www.blast.com

Table of Contents

Chapter 1

Introduction

BLAST Software Registration

Thank you for buying our communications software and welcome to the world of BLAST. Before doing anything else, it is *very* important that you complete the Warranty Registration Card. Without it, we cannot provide you with the complete support and continued service that comes with every copy of BLAST.

The services available to registered owners of BLAST include:

◊ A ninety-day warranty stating that the software will operate according to specifications in effect at the time of purchase.

◊ Professional help from our experienced Technical Support staff for a nominal fee.

◊ New product announcements.

◊ Discounts on product upgrades.

Extended warranties, custom support, special training, and corporate licensing are also available. Please call BLAST, Inc. at (919) 542-3007 or refer to the enclosed literature for more information.

The BLAST Package

The BLAST package contains the following items:

◊ One media package containing the BLAST program and support files.

◊ One BLAST Professional License Agreement and Warranty. It is important to read and understand the terms and conditions in this document before opening the media package.

◊ One Warranty Registration Card. The serial number of your BLAST program is printed on this card. When placing a call to BLAST Technical Support, please have this number available. Also, please read the card, fill it out, and send it immediately to BLAST, Inc.

◊ One Installation Guide and one User Manual.

If the package does not contain all of these items, please call the BLAST Customer Support staff.

BLAST Professional Features

BLAST Professional is designed to connect your computer with a variety of other computers. You may use one of the following connections:

◊ Communications devices such as modems, X.25 PADs, or ISDN Terminal Adaptors, and other virtual asynchronous circuits attached to RS-232 ports.

◊ Hardwired RS-232 connections.

◊ Telnet sockets.

◊ Raw TCP/IP sockets.

BLAST transfers files to and from remote computers with the fast, 100% error-free BLAST Session protocol. Alternatively, you may choose from one of the following protocols: Xmodem, Ymodem, Zmodem, or Kermit. BLAST Professional UNIX 10.8x users may also choose FTP.

The BLAST scripting language is a powerful but simple-to-use programming language. It allows the automation of communications tasks. Creation of scripts is simplified by the Learn mode feature of BLAST. Activate Learn mode and let BLAST write the script for you as you perform a communications task!

All UNIX products support TTY and PASSTHRU modes, which ensure complete and accurate transmission of control characters to the terminal hardware. BLAST Professional UNIX 10.7x supports several terminal emulations, including VT52, VT100, VT220, WYSE 50, WYSE 60, and ANSI.

10.7x also supports keyboard mapping and remote control that allows your computer to take complete control of a remote PC. Remote control works over modems and includes automatic translation between different video modes, password-protected dial-back security, and many other features.

How to Use This Manual

Parts of the Documentation System

Each portion of the BLAST documentation system fulfills a specific need:

◊ Online Help is always available while you are using BLAST. It is context-sensitive so that the information you need is right at hand.

◊ The Installation Guide contains step-by-step instructions for installing and configuring BLAST.

◊ The User Manual contains all the information necessary for operating BLAST, including detailed descriptions of Terminal mode and filetransfer procedures. It also contains general information as well as a listing of all BLAST functions, BLASTscript reserved variables, and BLASTscript statements. The listing for each BLASTscript statement includes syntax, usage details, and examples.

Documentation System Conventions

To help reduce confusion, BLAST documentation shares several common name conventions, display conventions, and defined terms:

◊ Examples in the text indicate the actual keystrokes you should type to perform a function. For example,

```
send myfile.txt ENTER
```

instructs you to type "`send myfile.txt`" and then press the ENTER key. In early introductory chapters, "ENTER" is included to indicate the keystroke needed to execute input of typed data. In later chapters, it is assumed and omitted.

◊ Italics in code indicate that the item (for example, a command line argument or a string value) is generic and that a more specific item is needed. For example, in the following lines of code,

```
Connect
Filetransfer
Send
local_filename
remote_filename
to
esc
```

specific filenames should be given for *`local_filename`* and *`remote_filename`*. An exception to this convention is the all-italic format used for command descriptions in Chapter 15.

◊ Differences between BLAST Professional UNIX 10.7x and BLAST Professional UNIX 10.8x and features unique to one or the other are indicated in headings and tables throughout the manual.

◊ The term "local" computer refers to the machine closest to you, whereas "remote" computer refers to the system to which your local machine is connected.

◊ The term "interactive" describes BLAST operation from the keyboard. When operating interactively, a user presses keys to control the program. Alternatively, a user may write a BLAST script to control the program.

◊ Finally, "Terminal mode" describes BLAST operation as a terminal to a remote computer. For example, if you are going to use BLAST to connect to a remote UNIX system, then your keystrokes will be interpreted by the remote computer as if you were operating from an attached terminal.

Comments and Suggestions

Considerable time and effort have been spent in the development of this product and its documentation. If you are pleased, or not pleased, we would like to hear from you. Please see the pages following the index of this manual for response forms that you may fill out.

BLAST Technical Support

If you have problems installing or running BLAST, first look for answers in your manual, particularly Appendix C, “Troubleshooting,” and in the Online Help. Double-check your communications settings, operating system paths, modem cables, and modem power switches.

If you are still unable to resolve the problem, contact BLAST Technical Support. For a nominal fee, a technician will help you with your problem. Technical Support may be purchased on a per-incident basis or annually. Contact our Sales Staff for details. If you purchased BLAST outside of the USA, please contact your authorized distributor for technical support.

What You Will Need To Know

Before you contact us, please have the following information ready:

◊ Your BLAST version number and serial number. These numbers appear in the opening banner (when you first start BLAST), in the Online Help window, and on your distribution media.

◊ Your operating system version number. To display your version number, type `uname -a` at the command line.

How to Contact Us

Telephone support is available Monday through Friday. If voice support is inconvenient, you may FAX your questions to BLAST, 24-hours-a-day. Please see the title page of this manual for contact numbers and the pages at the end of the manual for a sample FAX cover sheet.

Chapter 2

The BLAST Environment

Introduction

Multi-user environments are inherently complex. BLAST must work smoothly with all peripheral equipment and with other software programs loaded on your system. To help you integrate BLAST into your system, a set of environment variables and command line switches can be specified that customize the operation of BLAST. These features are described in this chapter in addition to a general discussion of communications ports and flow control.

Environment Variables

When BLAST is installed, by default all BLAST files are placed in the same directory, but you may choose to move the files to separate directories. Within BLAST, there are three different types of files and a separate environment variable pointing to the directory containing each type:

◊ *executable files* – program files with execute permission; the PATH environment variable points to the directory containing these files.

◊ *support files* – files required for normal operation of the software, including access to Online Help and the modem control library; the BLASTDIR environment variable points to the directory containing these files. *BLASTDIR must exist in order for BLAST to execute.*

◊ *auxiliary files* – setup files; the SETUPDIR environment variable points to the directory containing these files. If no SETUPDIR exists, BLAST will look to the BLASTDIR for setup files. Other files, such as script files, may reside in any directory of your file system.

Each user must have these environment variables set correctly. Typically you would edit each user's .profile, .login, or .cshrc to reflect this information.

Setting PATH, BLASTDIR, and SETUPDIR

To update your path temporarily and set the BLASTDIR environment variable, log in as a regular user and type the following at the shell prompt:

C Shell

set path=($path *executable_dirname*)
setenv BLASTDIR *support_file_dirname*
setenv SETUPDIR *auxiliary_dirname*

Bourne Shell and Korn Shell

PATH=$PATH:*executable_dirname*
BLASTDIR=*support_file_dirname*
SETUPDIR=*auxiliary_dirname*
export BLASTDIR SETUPDIR

where *executable_dirname* is the full path of the directory in which the BLAST files are stored, *support_file_dirname* is the full path of the directory in which the support files are stored, and *auxiliary_dirname* is the full path of the directory in which the auxiliary files are stored.

For example, if the executable and support files are in /usr/blast and the auxiliary files are in /usr/john, under the Bourne/Korn shells you would type:

```
PATH=$PATH:/usr/blast
BLASTDIR=/usr/blast
SETUPDIR=/usr/john
export BLASTDIR SETUPDIR
```

NOTE: This is only a temporary change. To set these values permanently, add the above commands to your system login procedure.

Additional Environment Variables

BLAST recognizes a number of additional environment variables for customizing its operation. The information in bold and brackets indicates the default value. Examples use the Bourne shell syntax. As with BLASTDIR and SETUPDIR, these environment variables must be exported.

BANNERTIME=*delay* 0 – 99 **[5]**

where *delay* is the time in seconds that the initial screen is displayed.

EXAMPLE:

BANNERTIME=2

BLASTDIR=*dirname* [/usr/blast]

where *dirname* is the directory that contains the BLAST support files such as systems.scr, modems.scr, blast.tdf, and blast.hlp. *BLASTDIR must exist in order for BLAST to execute!*

EXAMPLE:

BLASTDIR=/usr/blast

BPRINTER=*drivername* [/dev/lp]

where *drivername* is the target for printer output; BPRINTER can be set to a device or a print spooler.

EXAMPLE:

BPRINTER="lp -c %s >/dev/null"

This will cause BLAST to issue the lp command, substituting the print filename for %s.

EDITOR=*filename* [vi]

where *filename* is the name of the editor program that will be invoked by the Edit command from the Local menu. The default is the program vi, which must be located in your path.

EXAMPLE:

EDITOR=vi

SETUPDIR=*dirname* [$BLASTDIR]

where *dirname* is the directory in which the BLAST setup files are stored. The default SETUPDIR is the same directory as BLASTDIR. If many different users need to access BLAST, you may wish to point SETUPDIR to the $HOME directory of each user. Users can then maintain individual libraries of setup files. This technique permits the BLAST administrator to restrict access to the BLAST directory without limiting the ability of other users to run the software and create their own setups.

EXAMPLE:

SETUPDIR=$HOME

TERM=*terminal_name* [no default]

where *term_name* is the entry in the terminfo library that BLAST will use to control the terminal from which BLAST is being run.

EXAMPLE:

TERM=vt100

TMP=*dirname* [/usr/tmp]

where *dirname* is the directory in which temporary files will be stored.

EXAMPLE:

TMP=/usr/tmp

Command Line Switches

Command line switches allow you a number of options on startup. For example, you can automatically load a setup and run a BLAST script that brings you directly into a communications session without interactive input. BLAST recognizes the following switches and parameters. Some switches are for BLAST Professional UNIX 10.7x only and some are for 10.8x only (see descriptions of specific switches below):

blast [*setupname*] [-s*scriptname*] [*argument*] [-2] [-b] [-c] [-dd] [-dt] [-e] [-f] [-h] [-k] [-n] [-p] [-q] [-v or -?] [-x] [-y] [-z]

One space must precede each switch included on the command line. Do not insert a space between the switch and the parameter associated with it. For example, -s*scriptname* is correct, but -s *scriptname* is not.

setupname

specifies a setup file for BLAST to load. Note that it is not necessary to type the filename extension (.su). If a valid BLAST script is specified in the Script File field of the setup, the script will automatically execute (unless BLAST is started with the -h switch, in which case the script specified in the setup will be ignored). If no script is specified, BLAST will load the setup and display the Offline menu. If a setup is not specified on the command line, BLAST will automatically load the default setup (default.su). BLAST first checks for setups in the directory defined by SETUPDIR. If there is no SETUPDIR, BLAST checks the directory defined by BLASTDIR.

-s*scriptname*

specifies the BLAST script that will control the current session. Control will be passed automatically to the script instead of the regular BLAST menus and will return to the menu system at completion unless the script specifies that BLAST exit. If a script is named in the Script File field of the setup, the script specified by the -s option will override the one specified in the setup. Please note that no spaces are allowed between the -s and the script name. If no setup is specified on the command line, the default setup is loaded.

BLAST first checks for scripts in the current directory or in the path specified on the command line, then in SETUPDIR.

argument

Specifies one of ten optional arguments (text strings) that can be passed to a BLAST script directly from the command line. These arguments are stored as BLASTscript reserved variables `@ARG0` to

@ARG9. This option requires that a setup file be specified on the command line. If no setup is specified, BLAST will interpret the first argument as a setup name and will generate an error message if that setup does not exist.

10.7.5n	**-2** specifies four-digit format for year.

-b

forces BLAST to execute in batch mode, in which all displays are suppressed and the Local System shell is disabled. This switch allows BLAST to run with no output for batch operations. Local System commands executed from *within* BLAST may still generate output.

10.7x	**-c** forces BLAST to operate as if it were being run from a terminal instead of the computer console. Access mode is disabled when using this switch. This switch should be used if you experience problems running BLAST from the console of your system.

10.8x	**-dd** changes the default date format globally (see @DATEFORMAT on page 268). For example, -dd"%A:%B:%Y:%X" If the -dd switch is used on the same command line as the -y switch, the last switch on the line will take precedence. **-dt** changes the default time format globally. For example, -dt"%h:%m"

10.8x	**-e***number* specifies the end-of-transmission (EOT) timeout for Xmodem and Ymodem where timeout equals *number*/100 seconds. The minimum timeout is .1 second (`10`), and the maximum is 60 seconds (`6000`). For example, `-e1111` sets the timeout to 11.11 seconds. See Chapter 9 for more information on Xmodem and Ymodem. EOT timeout for Xmodem and Ymodem may also be specified with the BLASTscript reserved variable `@XYEOT`.

-f

enables XON/XOFF flow control for the port the user logs into.

-h

executes BLAST in host mode. In host mode, BLAST runs in Filetransfer and Answer mode connected through the port that is already open.

This command is usually issued from Terminal mode to start BLAST on a remote system. The remote system does not actually start BLAST protocol until the local computer begins file transfer. If the local system does not enter Filetransfer mode within the time specified in the Logon Time Out field of the remote setup, the remote computer will time out before logging on.

If used with an appropriately modified setup, the `-h` switch allows a local operator to change certain BLAST protocol parameters on the remote system temporarily. For example, if you had a remote setup called “special” that specified a packet size of `1024`, you could start BLAST with this parameter setting by specifying the setup “special” on the command line:

```
blast special -h
```

BLAST will look in BLASTDIR for this setup unless a SETUPDIR has been specified.

NOTE: In host mode, BLAST uses the login port parameters, ignoring the Script File setting and port parameters of the setup, except for XON/XOFF Pacing. See "Using BLAST in Host Mode" on page 26.

10.8x	Using the -h switch, BLAST can perform X, Y, or Zmodem file transfers. See "BLAST Operation as a Pseudohost With 10.8x" on page 204.

10.7x	**-k***country.kbd* loads an international keyboard driver, where *country.kbd* is the name of the driver. (See "International Keyboard for 10.7x" on page 29.

-n

forces BLAST to execute in no display mode. Displays may be selectively reenabled through BLASTscript commands. This switch allows you to integrate BLAST into your applications without losing the information previously written to the screen.

10.8x	**-p***x* specifies the pad character (*x*), expressed as a decimal value, to be be used with Xmodem transmissions. See Chapter 9 more information on Xmodem.

-q

forces BLAST into quiet mode. Audible signals that normally call attention to prompts and errors are suppressed.

-v or **-?**

displays the BLAST version, serial number, and command line switch usage.

-x

enables Extended Logging, which writes detailed information about BLAST protocol sessions to your session log. Extended Logging may also be enabled with the BLASTscript reserved variable `@XLOG`.

10.8x	**-y** specifies four-digit format for year. If the -y switch is used on the same command line as the -dd switch, the last switch on the line takes precedence.

10.7x	**-y***number* specifies the end-of-transmission (EOT) timeout for X and Ymodem where timeout is equal to *number*/100 seconds. The minimum timeout is .1 second (10) and the maximum is 60 seconds (6000). For example, -y1111 sets the timeout to 11.11 seconds. See Chapter 9 for more information on Xmodem and Ymodem.

-z

forces BLAST to attempt to open the communications port without changing the port's status. BLAST will not disable getty or ttymon processes automatically (see "Accessing Serial Ports" on page 18).

Example Command Line

The example command line shown below starts BLAST with a setup named "dial," a script named "newyork," and "30,400" as an argument to be used by the script, type:

```
blast dial -snewyork 30,400
```

Precedence for Specifying Options

Because the command line can specify options that may also be named in setups and scripts, BLAST follows a well-defined order of precedence:

◊ Whenever a command line switch conflicts with a value specified in a setup also loaded from the command line, the command line switch overrides the setup value.

◊ Whenever a command line switch conflicts with a setup value that has been loaded after starting BLAST (through interactive command or BLASTscript control), the setup value overrides the command line switch.

◊ Whenever a BLAST script changes a value specified in either the setup or the command line, the script change overrides the setup/command line value.

BLAST can establish a communications session with asynchronous serial ports and TCP/IP socket services. The port that BLAST will use is specified in the Connection setup field (page 69) or in the BLASTscript reserved variable `@COMMPORT` (page 266).

BLAST does not communicate directly with the computer hardware; rather, it accesses the hardware through a device driver. The device driver is a character-special device file usually found in the /dev directory. For network connections, BLAST talks to TCP/IP socket services.

In addition to device drivers, devices such as multi-port serial boards, terminal servers, and X.25 PADs permit software, like BLAST, to access the hardware. If the manufacturers of these devices do not provide a standard asynchronous interface, BLAST cannot open the device. If RS-232 capabilities are not correctly implemented in the device driver, those features will not be available during a BLAST session. For example, many drivers do not correctly implement modem control signals like DTR, DCD, RTS, and CTS.

Accessing TCP/IP Ports

BLAST makes network connections using TCP/IP socket services. Networking services must be correctly configured on your system for BLAST to make a network connection successfully, including having the host name in the/etc/hosts file or using Domain Name Services. Connecting to a specific port number requires that the port number be found in the /etc/services file. For more information on these configuration files, consult the hosts and services man pages.

Suppose that you have a modem connected to port 3001 on a terminal server called ts01. The terminal server's name is resolved via the hosts file or Domain Name Services, while the port number is in the services file. You can connect to that port and access the modem by entering

```
ts01 3001
```

in the Connection setup field (page 69). The port number should be separated from the host name by a single space.

NOTE: Port number 23 is reserved for telnet. To use telnet, simply enter the host name, and BLAST will default to port number 23.

To use telnet with a port other than port 23, enter the host name, the port number, and “telnet,” as in the example below:

```
blaster.blast.com 12 telnet
```

X.25 Communications and PADs

X.25 is a communications standard for transmitting data over packet switching public data networks. Public data networks provide long distance networking capabilities to users whose needs are not extensive enough to justify dedicated equipment and phone circuits. The interface to the public data networks is a PAD, which stands for Packet Assembler/Disassembler. If the PAD is directly attached to a system bus, the PAD manufacturer must provide a device driver that BLAST can open and set like a serial port. If the PAD is accessed through a modem or a standard serial port, BLAST can communicate with the PAD via a standard serial port device driver.

A PAD takes the data stream from a terminal or computer and assembles it into fixed length packets for transmission on a public data network. At the remote site, the packets are disassembled by the remote PAD and restored to the same form as the original data stream. A packet is transmitted when:

◊ Enough characters have been accumulated to form a complete X.25 packet. For many PADs, the default packet size is 128 bytes. Packet size is a modifiable PAD parameter.

◊ A “data-forwarding character” is encountered in the data stream. For many PADs, the default “data forwarding character” is a carriage return. This is a modifiable PAD parameter.

◊ A certain amount of time has expired without receiving a new character. The idle timeout period is a modifiable parameter. For interactive usage, the idle timeout should be set to a small value in order to improve “responsiveness.” This may, however, increase the number of partially empty packets.

The BLAST protocol is inherently compatible with X.25 communications: the BLAST packet size can be tuned to fit within an X.25 packet; by default, each BLAST packet is terminated with a carriage return; and the sliding window design of the BLAST protocol ensures that data is constantly being transmitted.

Optimum BLAST Packet Size

To operate efficiently over an X.25 network, BLAST protocol packet size must be optimally configured. The Packet Size setup field (page 90) or the reserved variable @PAKTSZ (page 279) specifies the number of bytes of data that BLAST will transmit in each BLAST packet. This specification does not include any bytes associated with BLAST's encoding of data, packet headers, launch characters, and CRC characters.

To make most efficient use of the X.25 connection, a BLAST frame—the data and the bytes associated with packetizing the data—must fit within the X.25 frame size. If the BLAST frame is too large to fit into a single X.25 frame, you will be sending a full frame and a partial frame. If the BLAST frame is too small, you will be sending partial X.25 frames.

There is a simple formula to determine optimal BLAST packet size for a given X.25 frame size. If you are using the 8-bit channel setting in the BLAST Protocol subwindow of the setup, the formula is:

$$\text{BLAST Packet} = \frac{[(\text{X.25 Frame} - 4) \times 7] - 9}{4}$$

If you are using the 7-bit channel setting, the formula is:

$$\text{BLAST Packet} = \frac{[(\text{X.25 Frame} - 5) \times 3] - 9}{4}$$

For example, if you are using a X.25 frame size of 256 bytes and an 8-bit channel, the optimal BLAST packet size is 219.

$$219.4 = \frac{[(256 - 4) \times 7] - 9}{8}$$

PAD Parameters

The X.3 standard specifies a set of parameters defining how the PAD is to perform its task of assembling and disassembling the data stream. The PAD must be properly configured for optimal performance. Please see Appendix F for a complete explanation of PAD Parameters.

Accessing Serial Ports

UNIX System V Release 3 and System V Release 4 use different methods for serial port configuration and control. The following sections discuss the two methods.

System V Release 3

For System V Release 3 (SVR3), you need to be familiar with the system programs init, getty, and login, and with the inittab configuration file. These programs operate in a loop called the IGLS cycle, init-getty-login-shell. In general, any attempt to control a serial port except through the IGLS cycle breaks the system's control of its resources. There is no provision built into the UNIX system to handle anything other than login terminals on serial ports. This can significantly affect the operation of BLAST.

The IGLS Cycle

The init process, process number 0 or 1, runs all the time. Periodically, it reads the file /etc/inittab to see if anything is pending. If the current UNIX system run level is 2 or 3 (UNIX is in a multi-user mode), and there is a line in inittab such as:

```
tty0:23:respawn:/etc/getty /dev/tty0
```

init will start a getty process using serial port /dev/tty0. The first field in the line, called a tag (tty0 in this example), is used as an arbitrary index into the inittab file. Tags must be unique.

getty is a simple process that reads /etc/gettydefs to find out how to configure serial port parameters, such as data bits, parity, and flow control, and then waits for activity on the port. As soon as the appropriate signal is received (usually a carriage return), getty starts a login process and then exits.

After verifying a username and password, login consults /etc/gettydefs to set line parameters and then starts a shell program. The serial port is now said to be the "control terminal" for the shell started by login. Finally, login informs the UNIX kernel of its actions and then terminates.

The shell program (e.g., sh and csh, etc.) is more properly called a command line interpreter and is capable of starting other programs, such as BLAST. When the user has finished and wants to exit the shell, an end-of-file signal (EOF) is sent. Exiting the shell notifies the UNIX kernel that the shell's control terminal is no longer in use. Consequently, the kernel sends a message to init telling it that /dev/tty0 has been released. Init then reads /etc/inittab and the IGLS cycle repeats.

Breaking the IGLS Cycle

Within the IGLS scheme, there is no provision for processes to initiate outbound connections. Even the programs UNIX-to-UNIX-copy (uucp) and Call UNIX (cu), long a standard part of UNIX distributions, must break the IGLS cycle to perform the tasks for which they were written.

There are two ways to handle this situation. First, it may be possible to dedicate some serial ports for outbound connections and reserve others for the IGLS cycle. This is the best solution, but it requires at least two serial ports, two modems, and possibly two phone lines to implement correctly.

Alternatively, it is possible to share a given serial port for dial-in and dial-out processes. The IGLS cycle can be disabled on a serial port by making a simple change to inittab:

```
tty0:23:off:/etc/getty /dev/tty0
```

where off replaces respawn. This change tells init not to use /dev/tty0. As soon as init examines the inittab file, it will shut down a getty process that is running on that port. The port is now available for uses other than terminal login. When the alternate process is finished, it can reenable the serial port for logins by restoring the original inittab entry.

As straightforward as this solution sounds, it poses a problem for the system because /etc/inittab is a system-level configuration file. If a user is permitted to start or stop the IGLS cycle on any serial port at any time, the integrity of the multi-user environment can be easily compromised. A number of mechanisms have been created to deal with this dilemma. For example, uugetty tries to distinguish automatically between inbound and outbound connections, setting up the serial port accordingly. For a number of reasons, however, it is difficult to make this configuration work reliably.

Another approach is to create "lock" files in a special directory and require programs wanting to use a serial port to check for the presence of a lock on the port before proceeding. Unfortunately, there is no way to enforce this procedure. Some programs do not check for locks at all, others do not interpret the lock information correctly, and there is no universally accepted location for the lock file directory. This issue is discussed in more detail in the section on "Port Locking" (see next page).

System V Release 4

System V Release 4 (SVR4) also attempts to control competition for serial ports. The IGLS cycle is essentially the same as in SVR3, but the functions of init and getty are now controlled by the port monitor daemon ttymon. Ttymon can monitor several ports, simplifying port administration. The port monitor uses configuration information stored in an internal database managed by the Port Monitor Administration (pmadm) program. Serial ports are enabled or disabled through commands to pmadm, while the port monitor itself is manipulated through commands to sacadm (the Service Access Control Administration program). These commands, pmadm and sacadm, are members of a set of commands for handling both network and terminal connections, collectively called the Service Access Facility (SAF).

Under SVR4, a serial port is disabled by issuing a command to pmadm in the format:

pmadm -d -p pmtag -s svctag

where -d is the disable option and pmtag and svctag are identifiers for the serial port. The output of the command pmadm - l will display these tags in addition to whether or not ttymon is monitoring a particular port:

```
PMTAG PMTYPE SVCTAG FLGS ID    <PMSPECIFIC>
zsmon ttymon ttya   u    root /dev/term/a I - /usr/bin/login - 9600
zsmon ttymon ttyb   ux   root /dev/term/b I - /usr/bin/login - 9600
```

An “x” appears under the FLGS column when ttymon is not monitoring a particular port. In the example above, ttymon is running on /dev/term/a but not on /dev/term/b. The pmtag for /dev/term/a is zsmon, and the svctag is ttya. Thus, the following command disables the port:

pmadm -d -p zsmon -s ttya

To enable the port, you would type:

pmadm -e -p zsmon -s ttya

Unfortunately, ttymon may not always release the port gracefully. On some systems it may be necessary to kill the ttymon process (through sacadm) or reboot before the port is flushed completely.

Port Locking

BLAST uses two of its own processes, setgetty and ttymgr, to manipulate system files directly and thereby lock and unlock serial ports. When the user goes online through the BLAST software, BLAST checks to see if the device specified in the Connection field of the setup is a character-special device. If it is, BLAST calls setgetty with information about the serial port, the user's UID, and other parameters.

setgetty and ttymgr

Setgetty first checks for a file called blasttab in the BLAST directory. If blasttab exists and is readable, setgetty will only allow access to ports listed in the file. The device named in the Connection field of the setup must match one of the entries in blasttab. After other checks are completed, setgetty calls ttymgr, the program that actually does the work of enabling and disabling the port. Owned by root, ttymgr runs with its set-uid bit enabled. It either modifies /etc/inittab (SVR3) or calls pmadm (SVR4).

If an error occurs, BLAST will display the message "can't open the communications port." In the event of an error, you can examine the contents of /usr/tmp/ttymgr.log to determine the cause of the problem. A complete description of the log format is beyond the scope of this manual. If you need to call BLAST Technical Support to help solve the problem, the support staff may need information contained in the log. If the port is successfully opened, BLAST resumes execution at the Online menu, and the user can select Connect, Terminal, or other Online functions.

Format of blasttab

The presence of blasttab, allows BLAST to prevent users from opening unauthorized ports on the system; blasttab must exist in the BLAST directory and have read permission enabled. The format of the file consists of the full path of the character-special device name that the user is allowed to open, one entry per line. Text following a "#" is treated as a comment and will not be used by setgetty. Blank lines are not permitted in the file. In the following file for example,

```
# Table of Ports For Use With BLAST
#
/dev/tty00          # High-speed 28.8 modem
/dev/tty01          # Low-speed 2400 modem for use with
                    # old systems
# /dev/ttyA16 --- this port not available
#
# end of blasttab. Updated 03/21/96 by dcb
```

the comment character before /dev/ttyA16 makes the port unavailable. Because of the special nature of this file, users other than root should not have write access to it.

Lock File Conventions

When BLAST gains access to a communications port, it creates lock files in appropriate system directories to make the port unavailable to other processes. Locations and naming conventions for lock files vary among UNIX systems. The following table illustrates the variety of implementations; use the table as a guide for locating where locks are kept in your system.

Platform	Directory	Naming Convention	Example
SCO	/usr/spool/locks /usr/spool/uucp	LCK..ttyn	LCK..tty2a
AIX	/etc/locks	LK.lkmajmin	LK.000.029.000
Solaris	/var/spool/locks	LK.lkmajmin	LK.000.029.000

BLAST normally removes its lock when it terminates. If BLAST receives certain UNIX signals, however, it may not be able to delete the lock file before exiting. In that case, you or your system administrator should manually delete the lock file before restarting BLAST.

Problems with Port Locks

BLAST's port locking scheme is comprehensive, but some circumstances can defeat it. For instance, multiple device drivers may refer to the same physical device, such as:

```
crw-rw-rw-  1 root  sys  5,  1 Dec 12 16:29  ttyd1
crw-rw-rw-  1 root  sys  5, 97 Aug 12 1994   ttyf1
```

If inittab or ttymon refers to the port as *ttyd1* but the BLAST setup refers to the port as *ttyf1*, port locking has no effect and BLAST will probably fail. One way to handle this problem is to have an entry in blasttab for /dev/ttyd1 and none for /dev/ttyf1, which would force users to specify /dev/ttyd1 within BLAST.

Choosing a Serial Port for BLAST

There is no standard naming convention for serial port device drivers; UNIX vendors and add-on board manufacturers have devised their own schemes. To add to the confusion, some vendors provide separate device drivers for modem connections and terminal connections. For complete information, you must consult the documentation for your system. On the following page are some examples of serial port device drivers on an assortment of UNIX systems.

Operating System	Device Driver
ATT System 5.4	/dev/term/a
Data General DG/UX	/dev/tty0
DEC Digital UNIX	/dev/tty00
Hewlett-Packard HP-UX	/dev/tty00
IBM AIX (also SCO 3.2.4.x, SCO Xenix)	/dev/tty0
SCO Open Server 5.0	/dev/tty1A
Silicon Graphics IRIX	/dev/ttyd1
Sun Solaris 2.3 (or /dev/ttya, not /dev/cua/a)	/dev/term/a

Serial port device drivers must have read and write permission in order for BLAST to access the port. You can check the permissions of a device driver by using the ls -l command. For example, to check the permission on /dev/tty01, type the following:

```
ls -l /dev/tty01
```

You should see output similar to the following:

```
crw-rw-rw-  uucp  uucp  77,128  Aug 24 10:47  /dev/tty01
```

The output "crw-rw-rw" indicates this file is a character-special device that has universal read and write permissions.

If the port does not have these permissions, you can change them using the chmod command. To change permissions, log in as root and type the following:

```
chmod 666 /dev/tty01
```

For more information on permissions and using chmod, consult the chmod man page and "Permissions" on page 150.

NOTE: These device names are often merely links to the actual character-special files. Other equivalent links may exist in a given system.

Using Links

BLAST supports links to serial port device drivers. A link can insulate inexperienced users from complex device names. For example, to create a link named /dev/blast to the device driver /dev/term/a, type the following:

```
ln -s /dev/term/a /dev/blast
```

After creating the link, you can check its existence by typing:

```
ls -l /dev/blast
```

You should see output similar to the following:

```
lrw-rw-rw- 2 uucp  uucp 77,128 Aug 25 08:27  /dev/blast —>  /dev/term/a
```

To use the link named /dev/blast, enter /dev/blast in the Connection field of the setup instead of /dev/term/a. For more on creating and using links, consult the ln man page.

Posix Vs. Non-Posix Drivers

BLAST can only open device drivers that comply with POSIX committee recommendations for serial port device drivers. In particular, device drivers conforming to BSD specifications may not be successfully opened by BLAST. For example, the device driver /dev/cua/a on a Sparc station running Solaris 2.x cannot be opened by BLAST, but the driver /dev/term/a, referring to the same physical port, can be.

Automatic Serial Port Searching

BLAST features automatic port searching using a special “hunt file” to locate an available port. To use automatic port searching, specify the name of a hunt file (including path, if necessary) preceded by “<” in the Connection field of the setup. For example, if a hunt file called hunt.fil resides in the BLAST directory, a setting of

```
<hunt.fil
```

in the Connection setup field specifies that BLAST will search hunt.fil and open the first available port listed there.

When you enter the Online menu, BLAST:

◊ looks for the hunt file in the current directory, then in the directory specified by the BLASTDIR environment variable.

◊ tests each listed port in the order specified until an available port (enabled or disabled) is found.

◊ returns the port to its previous condition when you exit BLAST.

If the hunt file is not found, or if none of the ports in the hunt file are available, you will receive the "Cannot open communications port" error message.

Hunt File Format

The hunt file is a standard ASCII text file in the following format:

setting device modem_type baud_rate

where:

setting	is either *try this device* (1) or *bypass this device* (0). A setting of 0 effectively removes the device from the table.
device	is the port name.
modem_type	specifies the modem type in the same format used for the Modem Type setup field (page 70).
baud_rate	specifies the baud rate in the same format used for the Baud Rate setup field (page 71).

IMPORTANT: The hunt file may not contain any extra spaces or lines. This applies to both the beginning and end of the file.

For example, BLAST would test the devices /dev/tty1 and /dev/tty3 listed in the following hunt file:

```
1 /dev/tty1 MICROCOM 19.2
0 /dev/tty2 USRCour 9600
1 /dev/tty3 Intel 9600
```

BLAST will ignore the entry for /dev/tty2 because it is preceded by a "0". Note that hunt files serve a different purpose than blasttab, described earlier, which is used for port validation. When a blasttab file is used, its entries must match all ports in a hunt file that are expected to be available.

Special Considerations

Using BLAST in Host Mode

Serial port device drivers have many modifiable parameters. Having these parameters set correctly significantly affects filetransfer and terminal scrolling speeds. In Answer or Originate mode, BLAST reads your setup file and attempts to set the device driver parameters accordingly when you go online.

When you log into a UNIX system, the system sets serial port parameters according to values in the /etc/gettydefs file. When BLAST is run in host mode on that system (using the -h switch), it does not attempt to reset serial port parameters. This generally works well, but in rare circumstances it may be necessary to change the settings before BLAST is invoked. UNIX provides the command stty for this purpose.

Viewing Serial Port Parameters with stty

To view the serial port parameters for the port into which you are currently logged, type:

```
stty -a
```

You should see output similar to the following:

```
speed 38.4 bps;
eucw 1:0:0:0, scrw 1:0:0:0
intr = ^c; quit = ^|; erase = ^?; kill = ^u;
eof = ^d; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^q; stop = ^s; susp = ^z; dsusp = ^y;
rprnt = ^r; flush = ^o; werase = ^w; lnext = ^v;
-parenb -parodd cs8 -cstopb -hupcl cread -clocal -loblk -crtscts -parext
-ignbrk brkint ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl -iuclc
ixon -ixany -ixoff imaxbel
isig icanon -xcase echo echoe echok -echonl -noflsh
-tostop echoctl -echoprt echoke -defecho -flusho -pendin iexten
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3
```

Unfortunately, there is no standard format for the output of stty. The output may have a substantially different appearance on your system; nevertheless, a number of important parameters are defined consistently across UNIX implementations:

stty Parameter	Meaning
speed 38.4 bps	The speed is 38.4 bits per second.
intr = ^c	The interrupt key is set to CTRL C.
erase = ^?	The erase character is set to CTRL ?.
parenb (-parenb)	Parity is enabled (disabled).
parodd (-parodd)	Parity is odd (even).
cs8 (cs7)	Data bits is 8 (7).
cstopb (-cstopb)	Use two (one) stop bits per character.
ixon (-ixon)	XON/XOFF [^Q/^S] output control is enabled (disabled).
ixany (-ixany)	Any character (only ^Q) restarts output.
ixoff (-ixoff)	XON/XOFF [^Q/^S] input queue control is enabled (disabled).

Setting Serial Port Parameters with stty

Stty can also set serial port parameters. For example, to change data bits from 7 to 8, you would type:

```
stty cs8
```

It may be necessary to change some parameters for optimal performance; however, the modem and serial port must cooperate. If, for instance, the modem is configured for XON/XOFF flow control and you set the serial port for RST/CTS flow control, you will encounter performance problems.

IMPORTANT: It is possible to change serial parameters so that the port will no longer respond to your terminal! Before making changes, copy the current settings so that they can be restored if necessary. A UNIX environment variable is useful here:

```
OLD_STTY='stty -g'
```

To restore the old parameters, simply type

```
stty $OLD_STTY
```

For example, some UNIX systems are set on login for even parity, 7 data bits, and 1 stop bit. BLAST file transfers, however, proceed more quickly if an 8-bit data path is available. On such systems, it may be possible to change the port parameters temporarily and restore them after file transfer is finished, as in the following commands:

```
OLD_STTY='stty -g'
stty cs8 -parenb -cstopb
blast -h
stty $OLD_STTY
```

Using BLAST 10.7x under SCO UNIX OpenServer 5

Executing BLAST 10.7x under SCO UNIX OpenServer5 requires null mapping of the character stream. Enter mapchan -n from the command line or add mapchan -n to the user's .profile to allow the character stream to be handled without translation. Failing to set mapchan -n can cause display and functionality problems in BLAST when run under OpenServer 5. For more information, refer to the man page for mapchan.

Running BLAST Remotely from the Console Using 10.7x

In order to run BLAST in Access mode, the Special KBD Mode of BHOST running on the remote computer must be set to ON. Also, the NUM LOCK key on the console keyboard should be turned off for correct cursor control. BLAST will reenable NUM LOCK on the console if NUM LOCK is engaged on a remote PC during remote control sessions. For information on configuring BHOST, see the *BHOST User Manual*.

International Keyboard for 10.7x

BLAST Professional UNIX 10.7x supports international keyboards through the -k command line switch. The following international keyboard driver files were copied to your system during the installation process:

french.kbd	italian.kbd	spanish.kbd
german.kbd	uk.kbd	

To load an international keyboard driver, add the following switch to the command line:

-k*country.kbd*

where *country.kbd* is one of files listed above.

File Format

The format for these files is:

scan_code = base, shift, ctrl, ctrl_shift, alt, alt_shift

scan_code	the key number of the key to be changed
base	ASCII value of new KEY
shift	ASCII value of SHIFT KEY
ctrl	ASCII value of CTRL KEY
ctrl_shift	ASCII value of CTRL SHIFT KEY
alt	ASCII value of ALT KEY
alt_shift	ASCII value of ALT SHIFT KEY

EXAMPLE:

2 = 49, 33, 0, 0, 96, 96

Refer to the actual files for more information. Lines that begin with the “#” character are comments.

Flow Control

Flow control paces the data stream between computers to prevent the loss of characters from data overruns. In serial communications, the primary factor adversely affecting transmission speed is an incorrect flow control setting. It is crucial to pace the data stream properly between connected computers to maximize filetransfer and terminal scrolling speed.

When data is received more quickly than it can be processed, the serial port buffer fills up. When the buffer is full, the device driver must halt the flow of data. If the serial port is connected to a modem, for example, some form of signaling is required so that the port can halt the flow of data from the modem as the serial buffer approaches capacity. Likewise, the modem must be able to signal the port to stop sending data if its own buffers fill up.

RTS/CTS Pacing

The RTS/CTS Pacing setup field (page 72) and the reserved variable `@RTSCTS` (page 284) enable a form of flow control that uses the RS-232 signals Request-To-Send (RTS) and Clear-To-Send (CTS). This type of flow control is sometimes referred to as hardware or “out-of-band” flow control. When the setup field is set to `YES`, BLAST attempts to set the serial port device driver to use RTS/CTS flow control; `NO` disables RTS/CTS flow control.

Unfortunately, UNIX serial port device drivers do not implement RTS/CTS flow control uniformly. The RS-232 standard originally defined these control signals for uni-directional flow control only, which is not appropriate for controlling full-duplex or bi-directional data flow. Consequently, some UNIX systems implement uni-directional RTS/CTS flow control in conformance with strict RS-232 specifications, whereas other UNIX systems offer a bi-directional form of RTS/CTS flow control that is compatible with modern high-speed modems. To make matters more confusing, some systems do not support any form of RTS/CTS flow control at all.

A device driver that supports bi-directional RTS/CTS flow control turns on RTS when the serial port buffer is ready to receive data and turns it off when the serial port buffer is full. Likewise, the modem turns on CTS when it is ready to receive data from the device driver and turns it off when the modem buffers are full.

If the device driver supports uni-directional RTS/CTS flow control, it will turn on RTS when it is ready to transmit data to the modem. In response, the modem will turn on CTS to indicate that it is ready to accept data. Unfortunately, as “uni-directional” implies, this scheme only controls the flow of data from the computer to the modem. There is no way for the device driver to halt the flow of data from the modem.

If you experience problems using RTS/CTS flow control, you should consult your system documentation (for example, man pages for stty and termio) to learn how RTS/CTS flow control is implemented in the device driver.

BLAST operates with greatest efficiency using bi-directional RTS/CTS flow control. If it is not available on your system, XON/XOFF flow control is the next best alternative. We do not recommend attempting to use uni-directional RTS/CTS flow control.

XON/XOFF Pacing

The XON/XOFF Pacing setup field (page 72) and reserved variable @XONXOFF (page 299) enable flow control based on the ASCII DC1 (XON) and DC3 (XOFF) characters. This type of flow control is often referred to as XON/XOFF flow control, software flow control, or “in-band” flow control. When the setup field is set to YES, BLAST attempts to set the serial port device driver to use XON/XOFF flow control; NO disables XON/XOFF flow control.

XON/XOFF flow control paces the flow of data by transmitting “start” and “stop” characters in the data stream. For example, when a modem receives an ASCII DC3 character, it stops transmitting data to the computer. When the modem receives an ASCII DC1 character, it restarts data transmission. This is analogous to starting and stopping terminal scrolling by pressing the CTRL S (XOFF) and CTRL Q (XON) keys.

XON/XOFF flow control is the most widely used form of flow control and is generally quite reliable; however, there are some potential problems:

◊ The protocol must not use the ASCII DC1 and DC3 characters to transmit data. Because Xmodem and Ymodem protocols use these characters, XON/XOFF flow control should not be used with these protocols. BLAST, Kermit, and Zmodem protocols *are* compatible with XON/XOFF flow control.

◊ The starting and stopping of data does not happen in real-time. Because the XON/XOFF characters are transmitted in the stream of data, there may be a substantial delay from the time when the XOFF is issued and when it is received by the transmitting device. This can cause data loss if buffers are overrun while the XOFF is being transmitted.

◊ If the XON character is lost, a protocol must implement a procedure to restart transmission or the file transfer will be irrevocably halted. The BLAST protocol, for example, will reset the device driver to begin data transmission if it does not receive an XON within 30 seconds of receiving an XOFF.

◊ In complex communications environments, it is possible to have many different pieces of equipment attempting to control data flow. For example, the device driver for the serial port, modems, terminal servers, and X.25 PADs can all be configured to assert flow control.

XON/XOFF flow control works most successfully in one of two ways, depending on the environment. In a simple environment, a local flow-control loop works best. In a more complex environment, an end-to-end flow control loop is most likely to work.

A local flow control loop is established when each modem is configured to act on the XON/XOFF characters sent by the attached computer. In this environment, the device driver will issue an XOFF when the serial port buffers are full. In response to the XOFF character, the modem will halt the flow of data to the computer. The modem will resume transmission when it receives an XON from the computer. Likewise, the modem will issue an XOFF to the serial port when its buffers are full.

The modems must be configured to act on flow control characters but not allow them to pass to the remote machine. No other devices should be configured to assert flow control. For best results, the modems should have an error-detecting connection established. BLAST does not recommend using local XON/XOFF flow control without an error-detecting connection. By default, when XON/XOFF pacing is enabled, BLAST establishes error-detecting flow control.

In more complex environments, or if error-detecting modems are not available, end-to-end XON/XOFF flow control should be used. In an end-to-end environment, the device driver will issue an XOFF when the serial port buffers are full. The XOFF character will pass through all devices to the remote computer, which will stop data transmission. When the buffers empty, an XON will be issued that causes the remote computer to restart the transmission. In similar fashion, if the remote machine's serial port buffers fill up, the device driver will issue an XOFF that causes the local machine to halt data transmission until an XON is received.

In this environment, all flow control should be disabled in the modems and all other equipment. You must manually configure the modems to do this or write your own entry in modems.scr (see "Sample Modem Script" on page 214).

Integration Options

This section discusses integrating BLAST with other applications. You need not read this section until you are familiar with the basic operation of BLAST described in the remainder of this manual. After you are comfortable using BLAST, you will find the following features very useful.

Shell Programming

BLAST is easy to run from UNIX shell scripts. In combination with the BLAST scripting language, many types of communications tasks can be automated.

The following example demonstrates running BLAST from a shell script. The example illustrates a Bourne/Korn shell; if you use another shell, please refer to your system documentation or other reference materials for more information. Many excellent references on shell programming are available and should be consulted for more complicated tasks.

A Basic Shell Script to Run BLAST

The script run_blast is a basic shell script for running BLAST:

```
# run_blast
# A demonstration script that sets environment variables, traps interrupts,
# runs BLAST, and checks the return code generated by BLAST.
#
trap '' 2                    # Trap any interrupts generated by BLAST on exit
                             # '' are two single quotation marks
PATH=$PATH:/usr/blast        # Append /usr/blast to the current PATH environ-
```

```
                                # ment variable.
BLASTDIR=/usr/blast             # Set BLASTDIR environment variable.
SETUPDIR=/usr/blast             # Set SETUPDIR environment variable.
#
export PATH BLASTDIR SETUPDIR   # Export PATH, BLASTDIR, and SET-
                                # UPDIR environment variables to sub-
                                # shells (Necessary in Bourne shell).
#
blast caller -stest.scr -b      # Run BLAST
#
RETURN_CODE=$?                  # Set RETURN_CODE environment
                                # variable to code returned by BLAST
#
if [ $RETURN_CODE -ne 0 ]       # Test for a nonzero return code
then
echo $RETURN_CODE               # echo code to screen if it is nonzero.
#
fi
#
# End of script
#
```

To execute the run_blast script, it is necessary to give it execute permissions. Run chmod by typing the following:

```
chmod 711 run_blast
```

This will set permissions on the script to read, write, and execute for the owner and execute only for group and other users. For some older systems, read permissions are required as well to run the script. For these systems, set the permissions as follows:

```
chmod 755 run_blast
```

Explanation

BLAST generates an interrupt when exiting. The line,

```
trap '' 2                  # '' are two single quotation marks
```

sets a trap for this interrupt, UNIX system Signal 2 (Terminal Interrupt). If this interrupt is not trapped, the shell script will terminate when BLAST exits, rather than continuing to completion. It is possible to set traps for other UNIX signals as well. For example, if you set a trap for Signal 15 (Software Termination), it will prevent the script from being terminated. To capture Signal 15 and echo “Please stop interrupting me while I am thinking” to the screen, include the following command in the script:

```
trap 'echo "Please stop interrupting me while I am thinking"' 15
```

The block of script,

```
PATH=$PATH:/usr/blast
BLASTDIR=/usr/blast
SETUPDIR=/usr/blast
#
export PATH BLASTDIR SETUPDIR
```

sets environment variables and exports them to subshells. In this example, the directory /usr/blast is appended to the search path the shell uses to find the BLAST executable. BLASTDIR and SETUPDIR are set to /usr/blast. For more information on environment variables and command line parameters used by BLAST, see “Environment Variables” on page 7.

The line,

```
blast caller -stest.scr -b
```

tells BLAST to load the setup caller.su, run the script test.scr, and execute BLAST in batch mode (no display). It is not necessary to run BLAST in batch mode from a script; however, if the script is going to be run in a non-attended mode or if it is going to run as a background process from cron, BLAST should always be run in batch mode.

The statement,

```
RETURN_CODE=$?
```

saves the return code from BLAST into an environment variable that can be used for further operations. BLAST will return a nonzero error code if an error occurs during processing (for a list of error codes, see Appendix A). It is also possible to use the BLAST scripting language to return error codes with the `QUIT` statement (for more on the `QUIT` script command, see page 250).

The following block of code tests the environment variable RETURN_CODE for a nonzero value. If RETURN_CODE is not equal to zero, the script will echo the return code to the screen.

```
if [ $RETURN_CODE -ne 0 ]
then
  echo $RETURN_CODE
fi
```

It may be useful to send the output of the return code to a log file along with a time stamp. For example, an environment variable called TIME_STAMP can be created by using cut to extract the time and date from the output of the date command by adding the following line to your shell script:

```
TIME_STAMP='date | cut -d " " -f 2-4'
```

The variable TIME_STAMP will contain information similar to the following:

Aug 21 11:16:56.

For more information, refer to the date and cut man pages. For more information on piping output from one command to another, refer to the man page of the shell you are using.

The time stamp and return code can be written out to the log file blast_error.log by including the following line in your script:

```
echo $TIME_STAMP "BLAST return code is: "$RETURN_CODE > blast_error.log
```

Running BLAST From cron

The need often arises to run BLAST on a regularly scheduled basis. A UNIX supplied clock utility, cron, executes jobs at specified dates and times, making it ideal for scheduling BLAST to run unattended. The following information on cron is applicable to many UNIX systems; however, cron may be implemented differently on your system. Please refer to your system documentation and the cron and crontab man pages for more information.

cron Primer

The cron utility is a daemon that runs when the system is booted. To be sure that cron is running on your system, type the following:

```
ps -ef | grep cron
```

This should generate output similar to the following line if cron is running:

```
root   260     1  0  Aug  3   ?       0:11 /etc/cron
```

If cron is not running, please consult your system documentation.

Jobs are usually submitted to cron with the crontab command. A crontab job is placed into the user’s subdirectory of the

/usr/spool/cron/crontabs directory; thus, root's crontab files will be in /usr/spool/cron/crontabs/root. It is also possible to edit a crontab file manually; however, cron only reads crontabs at the time of boot-up. A hand-edited crontab file will not be executed until the system is rebooted. Executing the crontab command will cause cron to re-read the crontab files immediately.

Not all users will necessarily be able to use crontab. On some systems access may be controlled by two files: /usr/lib/cron/cron.allow and /usr/lib/cron/cron.deny. If cron.deny exists and cron.allow does not, then all users except those listed in cron.deny will be able to access crontab. In this situation, if cron.deny is an empty file, all users will be able to use crontab. If neither file exists, only root will be able to use crontab. If cron.allow exists, only users listed in it will be able to use crontab. Please refer to the crontab man page entry for more information.

A crontab file is a standard text file with tab- or space-separated fields. The first five fields specify the minute (0–59), hour (0–23), day of the month (1–31), month of the year (1–12), and day of the week (0–6, with 0=Sunday). Each field can have one of the following:

◊ A number within the appropriate range for the field.

◊ Two numbers separated by a hyphen to indicate a range.

◊ A comma separated list of numbers.

◊ An asterisk to indicate all legal values.

The sixth field indicates the actual command to execute.

The following is an example of an actual crontab file:

```
17 5 * * 0 /etc/cleanup > /dev/null
0 2 * * 0,4 /usr/lib/cron/logchecker > /dev/null 2>&1
0 3 * * * /usr/lib/cleantmp > /dev/null
3,33 * * * 1-5 /usr/lib/uucp/uudemon.poll > /dev/null
```

The lines in this example perform the following actions:

◊ execute /etc/cleanup on Sunday morning at 5:17. Any output from cleanup is redirected to the bit bucket, /dev/null.

◊ execute /usr/lib/cron/logchecker on Sunday and Wednesday mornings at 2:00. Output is redirected to /dev/null. Error messages generated by logchecker are redirected to /dev/null as well.

◊ execute /usr/lib/cleantmp every morning at 3:00. Any output is redirected to /dev/null.

◊ execute /usr/lib/uucp/uudemon.poll at 3 minutes and 33 minutes past the hour every hour, Monday through Friday. Output is redirected to /dev/null.

BLAST under cron

Running BLAST from cron requires modifying your crontab file. To make a copy of your crontab file, type the following:

crontab -l > *my_cron*

This will make a copy of your current crontab file in *my_cron,* where *my_cron* is a valid filename. This copy of the crontab file can now be edited with any text editor. It would be a good idea to make a backup copy of the working crontab file prior to making any changes.

Your login configuration script (.profile, .login, .cshrc) will not be executed when cron executes your job; therefore, it is usually best to execute BLAST from a shell script that sets all of the necessary environment variables, including PATH, to run BLAST. Because cron runs in the background detached from any terminal, it is necessary to run BLAST with the -b switch in order to suppress screen output.

To execute the run_blast shell script (see "Shell Programming" on page 33) from cron at 2:30 a.m. Tuesday through Saturday, add the following line to the file *my_cron*:

30 2 * * 3-7 run_blast

To submit the new crontab file to cron, type the following:

crontab *my_cron*

Chapter 3

BLAST Quickstart

IMPORTANT: The following section assumes that BLAST has been properly installed. *Before* proceeding, be sure to:

◊ Successfully complete the entire BLAST installation process as instructed in the BLAST Installation Guide.

◊ Connect the modem according to the instructions supplied by the modem manufacturer and turn on the modem.

Starting BLAST

The command to execute BLAST is issued at the operating system prompt. Type:

blast

and press the ENTER key.

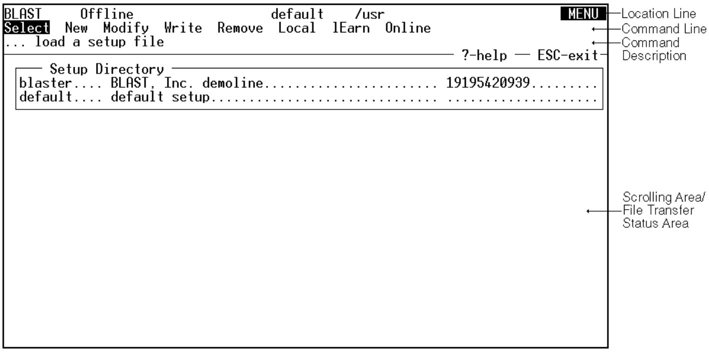
Make sure that all environment variables are set prior to executing BLAST. Consult “Environment Variables” on page 7, your system documentation, and the man page for your shell for more information.

If this is the first time you have run BLAST, the Online Help screen appears automatically (only on the first time). You can either explore the Help menu now or cancel it until needed. BLAST displays the Offline menu next. You can now control BLAST interactively.

The BLAST Screen

The BLAST screen (Figure 3-1) includes two sections: the Command Area and the Scrolling Area.

FIGURE 3-1



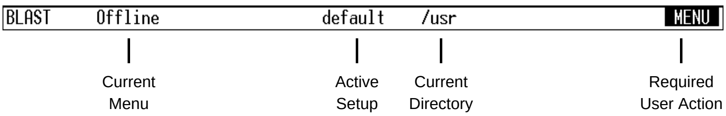
Command Area

The Command Area consists of three lines: the Location Line, the Command Line, and the Command Description.

Location Line

The Location Line provides the information about your “location” within BLAST (Figure 3-2):

FIGURE 3-2



Current Menu – displays the BLAST menu currently in use. The possible values are Offline, Online, Filetransfer, Local, and Remote.

Active Setup – displays the setup that is currently loaded.

Current Directory – identifies the current directory. Use the Chdir command in the Local menu to change the current directory.

Required User Action – displays the action that BLAST expects from you. Possible values are:

MENU – select a command from the menu.

INPUT – type in data at the prompt.

ERROR – review the error message, then press any key.

WAIT – no action allowed, BLAST is busy.

SCRIPT – a BLAST script is executing.

ONLINE – BLAST is online.

Command Line

The Command Line (Figure 3-1) lists the commands available from the menu.

Command Description

The Command Description (Figure 3-1) gives a one-line explanation of the command currently highlighted by the cursor. If you need more information about the command, press the *HELP* key (for more on the *HELP* key, see page 43).

Scrolling Region

The Scrolling Region is the area below the Command Area. Depending on the menu selection, this area is either blank or displays status and data. The format of the display depends on the activity BLAST is performing.

File Transfer Status Area

During file transfers, the scrolling area displays information about files being transferred. This display, called the File Transfer Status Area (Figure 3-3, next page), differs slightly depending on the protocol used.

Following is a description of each item, or status indicator, in the BLAST protocol File Transfer Status Area.

local – the name of the file that your system is sending or receiving.

opt – the optional transfer switches that you selected for this file.

%xfer – the percentage of the file that has been transferred to or from the remote machine.

file size – the total file size (in bytes).

byte count – the portion of the file that has been transferred to or from the remote machine (in bytes).

ln qual – a general description of the line quality of the connection between the computers. Possible values during a transfer are `good`, `fair`, `poor`, or `dead`.

Unlike BLAST protocol, other supported protocols do not make use of all the above status indicators.

FIGURE 3-3

```
BLAST     Filetransfer             default    /usr                        MENU
Send  Get  Message  Remote  Local  File
... send file(s) to the remote system
— local ——————————————————— opt – % xfer – file size —— byte cnt – ln qual –
S: <idle>
R: <message>                                                          good (00)
——————————————————————————————————————————————————————————— ?-help —— ESC-exit
-----------------------<< Entering BLAST Transfer Mode >>----------------------
****  MacBLAST 10.2 on remote system [uovm]  ****
```

Three Keys to Remember

A number of special keys are used within BLAST, but three are used frequently:

ATTN CTRL K is the default "Attention (*ATTN*) Key." Press CTRL K to abort script operations or initiate other special key combinations. Press CTRL K CTRL K to return to the Online menu from Terminal mode. (The *ATTN* key can be redefined; see "Attention Key" on page 73.)

CANCEL To cancel the current action, return to the previous menu, or exit BLAST:

10.7x	Press ESC.
10.8x	Press ESC or CTRL A.

HELP While in Terminal mode, press *ATTN* H,

10.7x	For context-sensitive Online Help when not in Terminal mode, press F1 from the main console or ? from attached terminals. While running BLAST, you can set *HELP* to any key using the keyboard mapping utility, blastkbd (see "Keyboard Mapping Utility for 10.7x" on page 315).
10.8x	When not in Terminal mode, press ?.

The BLAST Menus

Within menus, move from one command to another by pressing SPACEBAR or BACKSPACE.

10.8x	Alternatively, you may use the cursor keys to move from one command to another.

Select a command by pressing the capitalized letter in the command or by pressing ENTER when the cursor rests on the desired command. After opening a submenu, return to the previous menu by pressing *CANCEL*.

Below the menu is a one-line description of the current command (Command Description Line). To get more information, press the *HELP* key when the cursor highlights the appropriate command. After displaying text related to the command, BLAST displays a general Help section on other topics. See Chapter 4 for a detailed discussion of the menus.

Menu Summary

Each of the menus offers commands that are grouped together by function. For example, the Local menu allows you to manage your system while online with a remote system, whereas the Filetransfer menu provides functions connected with sending and receiving files.

Following is a brief summary of each major menu and its purpose:

Offline – Manages the setups that contain connection information.

Online – Manages connecting to and disconnecting from a remote system; executes BLAST scripts; sends and captures text files; and starts Terminal mode.

Filetransfer – Sends and receives files using either BLAST, Kermit, Xmodem, Ymodem, or Zmodem protocol.

10.8x	Also sends and receives using FTP.

Remote – Available with BLAST protocol and Kermit protocol. Performs file management on the remote system.

10.8x	Also available with FTP.

Local – Performs file management on your local system and provides access to the operating system command line.

A Quickstart File Transfer

The most common use of BLAST is communicating between two computers using standard asynchronous modems and ordinary telephone lines. BLAST provides "hands-on" experience in this environment through a computer system called Blaster. This system is available 24-hours-a-day, seven days a week for BLAST demonstrations and testing. You are encouraged to take advantage of this service to familiarize yourself with the many features of BLAST.

This section of Quickstart will guide you through:

◊ Selecting the Blaster setup.

◊ Connecting to Blaster.

◊ Performing BLAST protocol transfers.

◊ Logging off Blaster.

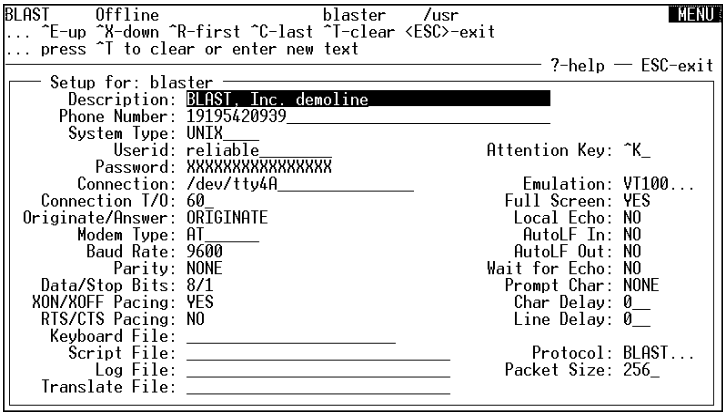
Although we recommend that you complete this section in one sitting, you may elect to stop by returning to the Online menu and choosing the Disconnect command.

Selecting the Blaster Setup

Setups contain all the information that BLAST needs to connect to and communicate with remote computers. Each setup is a separate file, created and modified through the Setup window of the Offline menu. This process is described in detail in Chapter 5. For this demonstration, you will use the setup called blaster.su, which was copied to your disk during the installation process.

If you have been moving through the menus, press the *CANCEL* key until you return to the Offline menu. Press S for Select and then press the ENTER key. You should see "blaster" listed as one of the entries in the Setup Directory. Use the keys listed at the top of the Command Area to highlight blaster and then press ENTER. You should see the Setup window shown in Figure 3-4 below.

FIGURE 3-4



Check to see that the following entries appear correctly on the screen:

Phone Number:	`1-919-542-0939`
System Type:	`UNIX`
Userid:	`reliable`
Password:	`XXXXXX` (`fast-transfer` is the actual password, but it will be masked by "Xs")
Parity:	`None`
Data/Stop Bits:	`8/1`
Emulation:	`VT100` or `PASSTHRU`
Protocol:	`BLAST`

If any of the entries are incorrect, press M for Modify and use the keys listed at the top of the Command Area to move to the appropriate field and enter the information. For the fields Phone Number, Userid, and Password, press CTRL T to clear the field and type in the correct information (remembering that the userid and password are case-sensitive) and press ENTER.

For the remaining fields shown above, you can cycle through the available choices by pressing the SPACEBAR. For the Emulation field, select PASSTHRU. For correct settings of the setup fields Connection, Modem Type, Baud Rate, XON/XOFF Pacing, and RTS/CTS Pacing, see your hardware documentation, your system administrator, and the discussion of these setup fields in Chapter 5.

After you are satisfied that all of the setup information is correct, press the *CANCEL* key to exit to the Offline menu. If you made any changes to the setup, press W (for Write) to save the changes.

Blaster assumes that a dial-in user will be using a VT-100 terminal. If you are using a VT terminal or your console operates as a VT or ANSI terminal, you should not have problems. Problems such as the screen not clearing, improper positioning of characters, and strange character sequences indicate that you are not using a VT terminal.

If your terminal is incompatible with a VT100, the best solution is to reset the TERM environment variable on Blaster to match the type of terminal you are using. For example, if you are using a WYSE 60 terminal, type the following after logging into Blaster:

```
TERM=wy60
```

This will cause Blaster to send WYSE 60 controls to your terminal instead of VT100 controls.

It may also be possible to reset your console or terminal to emulate a VT100. For example, if you are running BLAST from an xterm session and experience the problems described above, your xterm may be defaulting to an emulation other than VT100. On many systems, you can start xterm in VT100 mode by typing:

```
xterm +t
```

For more information on resetting your console or terminal, consult your system or hardware documentation. For more information on xterm, consult the xterm man page.

Connecting to Blaster

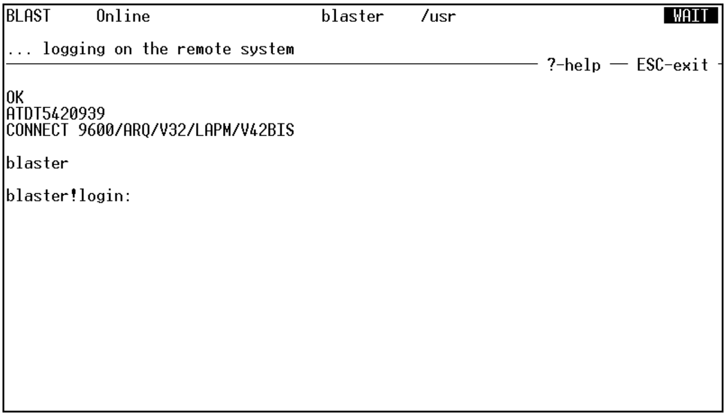
Your system is now ready to begin talking to Blaster. You have already loaded the blaster setup into memory with the Select command described in the previous section. Now press `o` to go to the Online menu. Connect to Blaster by pressing `c` (Connect) which will automatically dial Blaster.

The screen will display messages for each of the steps in the Connect process. If your modem has a speaker, listen to make sure it dials the number. Also, watch the terminal dialogue between the computer and the modem. When the call is successful, a message displays indicating that the connection has been established:

CONNECT nnnn

where *nnnn*, if present, gives connection information and speed (Figure 3-5).

FIGURE 3-5



After recognizing the modem's `CONNECT` message, Blaster's banner and request for login will be displayed. Your setup file will automatically enter the userid and password. When the login is complete, BLAST returns control to you by displaying the Online menu and waiting for your input (Figure 3-6, next page).

```
BLAST     Online                   blaster    /usr                       MENU
Connect  Terminal  caPture  Upload  Filetransfer  Script  Local  Access  Disc
... become a terminal to the remote system
──────────────────────────────────────────────────────────── ?-help ── ESC-exit ─

                        W E L C O M E  t o  B L A S T ,  I N C .

        You have logged into Blaster, our SCO Unix machine at BLAST, Inc.
        This login is for the use and convenience of customers of BLAST, Inc.
        Under this login, you can get sample scripts from the scripts
        directory and technical notes from the text_documents directory.

$
```

FIGURE 3-6

Performing BLAST Protocol Transfers

To begin file transfer, select the Filetransfer command from the Online menu by pressing F. In a moment, Blaster will synchronize with your system and display the Filetransfer menu (Figure 3-7).

```
BLAST     Filetransfer             blaster    /usr                       MENU
Send  Get  Message  Remote  Local  File
... get file(s) from the remote system
── local ───────────────────── opt ─ % xfer ─ file size ── byte cnt ─ ln qual ─
S: <idle>                                                              good
R: <idle>                                                              good (00)
──────────────────────────────────────────────────────────── ?-help ── ESC-exit ─
        You have logged into Blaster, our SCO Unix machine at BLAST, Inc.
        This login is for the use and convenience of customers of BLAST, Inc.
        Under this login, you can get sample scripts from the scripts
        directory and technical notes from the text_documents directory.

$
$ blast -h
;starting BLAST protocol
-----------------------<< Entering BLAST Transfer Mode >>----------------------
****  BLAST Professional UNIX 10.7.3 on remote system [uov]  ****
```

FIGURE 3-7

Getting a File from Blaster

To get a file from Blaster, select Get by pressing G. BLAST will prompt with:

enter remote filename:

BLAST is asking for the name of the file to retrieve. Type:

`blaster.msg` ENTER

BLAST will prompt with:

enter local filename:

A name may be specified for the incoming file. Type:

`news.msg` ENTER

BLAST will prompt with:

specify transfer options (t=text, o=overwrite, a=append):

To transfer this file using text format translation, type:

`t` ENTER

BLAST will begin retrieving the file, and the byte count in the File Transfer Status Area will increase.

After the file has been completely sent, the byte count will stop, a blank will appear in the byte count status indicator, and the following message will appear on your screen:

news.msg/T=TXT... receive completed

Sending a File

To send a file to Blaster, select the Send option by pressing `s`. BLAST will prompt with:

enter local filename:

Type:

`news.msg` ENTER

BLAST will then prompt with:

enter remote filename:

BLAST is asking for the name that will be given to the file when it is transferred to Blaster. Type:

`news.msg` ENTER

BLAST will prompt with:

specify transfer options (t=text, o=overwrite, a=append):

Because this is a text file, press T for text translation and O to overwrite any old versions of this file:

`to` ENTER

Again, notice that the status fields are updated as the file transfer progresses. At the end of the transfer, you will see the following line displayed on your screen:

news.msg/T=TXT news.msg/OVW/T=TXT... send completed

After the file transfer is complete, press *CANCEL* to return to the Online menu. An orderly shutdown of the BLAST protocol will follow and the Online menu will appear.

Logging off Blaster

Select the Disconnect command by pressing D. To quit BLAST, press *CANCEL* twice. BLAST will prompt with:

No Yes
...do you really want to leave BLAST?

Press Y to quit.

Chapter 4

The Menus

This chapter guides you through the various BLAST command menus. Some items covered here are described in more detail in other chapters; in such cases, you will be referred to the appropriate chapter. Each menu offers commands that are grouped together by function. For example, the Local menu allows you to manage your system while online with a remote system, whereas the Filetransfer menu provides functions connected with sending and receiving files.

Moving Through the Menus

Within the command line of a menu, you may move from one command to another by pressing SPACEBAR or BACKSPACE.

10.8x	Alternatively, you may use the cursor keys to move from one command to another.

Execute a command by pressing the capitalized letter in the command or by pressing ENTER when the cursor rests on the desired command. After opening a submenu, return to the previous menu by pressing *CANCEL*. For a discussion of selecting a setup and navigating through a Setup window, see "What is a Setup?" on page 63.

BLAST uses special key sequences to differentiate between local commands and characters meant for the remote system. The BLAST Keys perform local functions, such as exiting Terminal mode or displaying Online Help. BLAST Keys are most important in Terminal mode, when BLAST ordinarily sends all keystrokes directly to the remote computer. All of the BLAST keys are listed in Appendix B. In BLAST Professional UNIX 10.7x, some keys can be reassigned using the BLAST keyboard utility, blastkbd (see "Keyboard Mapping Utility for 10.7x" on page 315 for details).

Three Keys to Remember

You will use three of the BLAST Keys most often:

ATTN CTRL K is the default "Attention (*ATTN*) Key." Press CTRL K to abort script operations or initiate other special key combinations. Press CTRL K CTRL K to return to the Online menu from Terminal mode. (The *ATTN* key can be redefined; see "Attention Key" on page 73).

CANCEL To cancel the current action, return to the previous menu, or exit BLAST:

10.7x	Press ESC.
10.8x	Press ESC or CTRL A.

HELP While in Terminal mode, press *ATTN* H,

10.7x	For context-sensitive Online Help when not in Terminal mode, press F1 from the main console or ? from attached terminals. While running BLAST, you can set *HELP* to any key using the keyboard mapping utility, blastkbd (see "Keyboard Mapping Utility for 10.7x" on page 315 for details).
10.8x	When not in Terminal mode, press ?.

The Attention Key

The Attention Key alerts BLAST to prepare for a particular operation. The Attention Key is actually two keys, CTRL plus another character, represented in this documentation by the symbol "*ATTN.*" The default Attention Key is CTRL K. Press CTRL K to abort script operations or initiate other special key combinations. Press CTRL K CTRL K (*ATTN ATTN)* to return to the Online menu from Terminal mode. To transmit the control characters as *ATTN* to a remote system, press *ATTN* and then the character itself. For example, CTRL K K will transmit a CTRL K to the remote system.

You may change the default value of the Attention Key by altering the value of the Attention Key setup field (page 73) or by setting the BLASTscript reserved variable @ATTKEY (page 265).

NOTE: If it is necessary to change the Attention Key, be sure to choose a replacement value that will not interfere with your system's designated control codes. In particular, do not use CTRL M, which is the control code for a carriage return. Check your system manual for more information about special control codes *before* you reassign the Attention Key.

The Attention Key can initiate many useful functions from Terminal mode. Please refer to Appendix B for all of the Attention Key sequences.

The Cancel Key

The *CANCEL* key is used to cancel the current action. It also returns to a previous menu from a lower level menu and is used to exit BLAST from the Offline menu. The exception to this rule is that you must press *ATTN ATTN* to escape from Terminal mode.

The Help Key

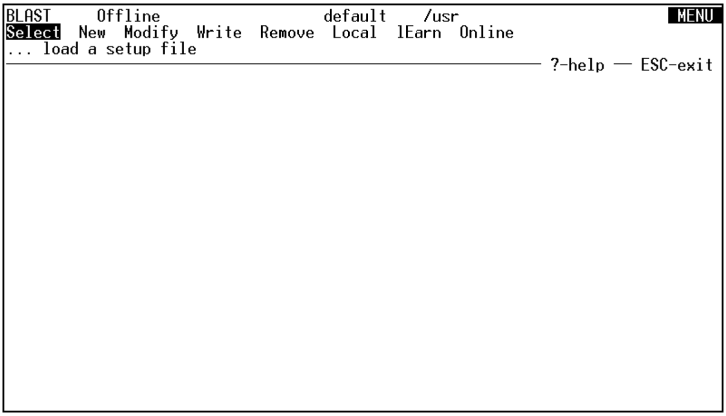
When the cursor rests on a command in the menu, pressing *HELP* will display Help about that particular topic. After displaying text related to the command, BLAST displays a general Help section on other topics.

Other Special Keys with 10.7x

Other special types of keys are available with BLAST Professional UNIX 10.7x. See "Keyboard Mapping Utility for 10.7x" on page 315 and Appendix B for details.

The Offline menu (Figure 4-1) is the first one displayed when you execute the BLAST program. The display includes three sections: the Command Area, the Scrolling Area, and the Status Line. See "The BLAST Screen" on page 40 for a description of these sections. We will be concerned here primarily with the Command Area, specifically the Command Line.

FIGURE 4-1



If this is the first time that BLAST has run, the Help screen will appear; press *CANCEL* to leave the Help screen. For Online Help, press the *HELP* key when the cursor rests on the appropriate command.

Setup Commands

Five of the commands in the Command Line of the Offline menu affect the setups listed in the Setup Directory and displayed in the Setup window (see "What is a Setup?" on page 63 more details).

Following is a brief description of each command.

Select – Displays an input field for entering the name of the setup file to load into memory. If you press ENTER while the field is empty, the Setup Directory will be displayed in the scrolling area. Use the SPACEBAR to highlight a setup in the directory and press ENTER to load it.

New – Prompts you for a new setup name. Type the name, press ENTER, and BLAST will automatically enter the Modify mode, displaying in the Setup window the values of the setup currently loaded in memory.

Modify – Displays in the Setup window the current values of the setup in memory and allows you to make changes. Upon exiting Modify mode, those values will be loaded into memory.

Write – Saves the current values in memory to the setup file named on the location line.

Remove – Prompts you for the name of a setup to delete. If you press ENTER while the field is empty, the Setup Directory will be displayed in the scrolling area. You can then use the SPACEBAR to highlight a setup in the directory and press ENTER to delete it.

Other Offline Commands

Local – Allows you to perform local system commands by taking you to the Local menu (described in detail on page 58).

Learn – Builds a script for you by starting Learn mode. When you execute the Learn command, you will be prompted for a script name. After you type the name and press ENTER, BLAST will record all of subsequent functions in the script file until you disable Learn mode by selecting the Learn command again. If you specify an existing filename for the script, BLAST will ask whether you want to append to or overwrite the original script file. See "Learn Mode" on page 176 for more details.

NOTE: Learn mode may not function consistently in PASSTHRU emulation.

Online – Takes you to the Online menu, described in the next section.

The Online Menu

Selecting Online from the Offline menu displays a menu like or similar to the one shown in Figure 4-2 (next page). All characters received and transmitted in Terminal, Capture, and Upload modes will be filtered by the translate file if one is specified in the Translate File setup field (page 73). See “Translate File Format” on page 306 for more information on translate files.

Following is a brief description of the commands of the Online menu.

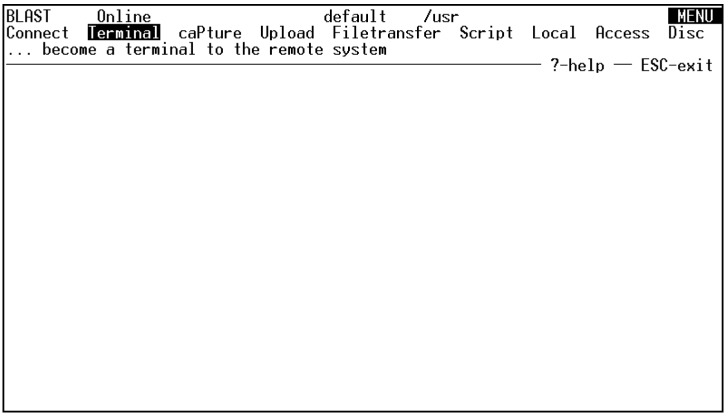
Connect – Dials the phone number stored in memory from the current setup.

Terminal – Makes your system a terminal to the remote system. The menu commands will no longer be available to you. Remember that you must press *ATTN ATTN* in order to exit Terminal mode and return to the command menus. See “Standard BLAST Terminals” on page 309 and “Terminal Emulation with 10.7x” on page 311 for further information.

Capture – Causes all incoming text from the remote system to be captured to a file. When you enter Capture mode by executing the Capture command, you will be prompted for a filename; type the name and press ENTER. BLAST will record in the capture file all subsequent text displayed in the Terminal window until you disable Capture mode by selecting the Capture command again. If you specify an existing filename for the capture file, BLAST will ask whether you want to append to or overwrite the original file. See “Downloading Text from a Remote Computer” on page 147 for further information.

Upload – Sends text from a local file to the remote computer and displays the text on your screen. See “Uploading Text to a Remote Computer” on page 145 for further information.

FIGURE 4-2



Filetransfer – Takes you to the Filetransfer menu described in the next section. See also chapters on individual protocols.

Script – Executes a BLAST script after prompting you to enter the script name. See Chapters 12–14 for information on scripts.

Local – Allows you to perform local system commands. This command takes you to the Local menu described on page 58.

10.7x	**Access –** Begins a remote control session. After entering Access mode, *ATTN* takes you to an Access menu (see “The Access Menu” on page 327).
	Disc – Logs off of the remote system cleanly and hangs up the modem using information from the System Type and Modem Type setup fields.
10.8x	**Disconnect –** Logs off of the remote system cleanly and hangs up the modem using information from the System Type and Modem Type setup fields.

The Filetransfer Menu

Selecting the Filetransfer command from the Online menu displays the Filetransfer menu. The Filetransfer menu for BLAST protocol is shown in Figure 4-3 below.

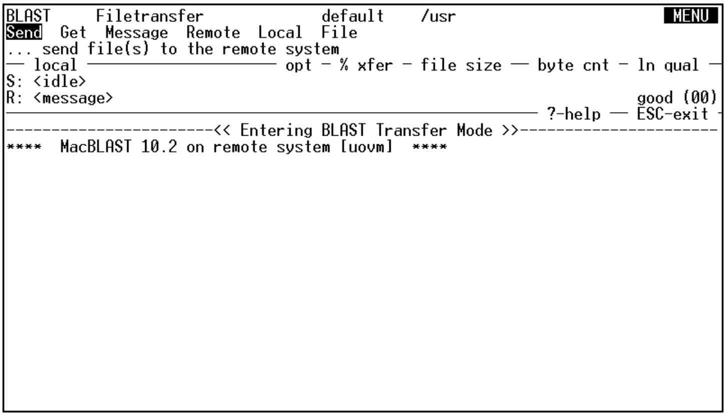

```
BLAST     Filetransfer            default    /usr                    MENU
Send  Get  Message  Remote  Local  File
... send file(s) to the remote system
── local ──────────────────── opt ─ % xfer ─ file size ── byte cnt ─ ln qual ─
S: <idle>
R: <message>                                                       good (00)
──────────────────────────────────────────────────────── ?-help ── ESC-exit ─
-----------------------<< Entering BLAST Transfer Mode >>---------------------
****  MacBLAST 10.2 on remote system [uovm]  ****
```

FIGURE 4-3

The commands on the Command Line of the Filetransfer menu vary depending on the protocol used. For example, the X, Y, and Zmodem protocols will only display the Get and Send commands, whereas the Kermit protocol has additional options and its own special Remote submenu. Following is a brief description of the commands of the BLAST protocol Filetransfer menu. For more information on menu options for protocols other than BLAST protocol, see chapters discussing individual protocols.

Send – Sends a file or files to the remote system.

Get – Retrieves a file or files from the remote system.

Message – Sends a message to the remote operator. Simply type the message and press ENTER. The message will be queued for transmission to the remote display.

Remote – Performs remote system commands allowing limited access to the remote computer. The BLAST protocol Remote menu commands, which are similar to the Local commands, are described on page 60; see also "FTP Remote Menu" on page 128 and "Kermit Remote Menu" on page 134.

Local – Performs local system commands. This command takes you to the Local menu, described in the next section. Note that all filetransfer activity is suspended while you are using the local system. This inactivity may exceed the interval specified by the BLAST protocol Inactivity Timeout setup field (page 85) and terminate Filetransfer mode.

File – Executes a transfer command file that can control an entire BLAST protocol transfer unattended (see "Transfer Command File" on page 115, "Transfer Command File" on page 361, and "Transfer Command Files" on page 364).

The Local Menu

The Local menu (Figure 4-4, next page) allows you to perform operations on your local computer, including escaping to a command shell. Local commands affect only files in the current directory unless you specify a pathname.

FIGURE 4-4

```
BLAST     Local                        default    /usr/customer              MENU
List  Delete  Edit  Rename  Type  View  Print  Chdir  System
... list filenames
--------------------------------------------------------------- ?-help — ESC-exit
total 698
-rw-r--r--   1 reliable group        0 Jan 01  1970
drwxr-xr-x   5 reliable group      112 Jan 18 05:33 EPTEST
-rw-r--r--   1 reliable group     1808 Apr 14 09:32 Initial.dt
-rw-r--r--   1 reliable group       23 Mar 07 17:17 XDesktop3
-rw-r--r--   1 reliable group     2000 May 30 10:44 agyaudit.3aid
drwxr-xr-x   2 reliable other      176 May 30 16:55 atlas
drwxr-xr-x   2 root     other       48 Mar 31 17:32 bin
-rw-r--r--   1 reliable group   187609 May 31 15:12 blast.log
-rw-r--r--   1 reliable group    22067 Mar 07 17:18 blaster.fil1
-rw-r--r--   1 reliable group    22067 Mar 07 17:18 blaster.msg
-rw-r--r--   1 reliable group    29184 Jan 01  1970 blaststp.exe
drwxr-xr-x   2 reliable group       96 May 29 03:20 cho
drwxr-xr-x   2 reliable group       32 Apr 25 08:36 clipdir
drwxr-xr-x   2 reliable group      160 May 26 09:42 cosyda
-rw-r--r--   1 reliable group     1200 May 30 10:41 from.053095
drwxr-xr-x   2 reliable group       64 Dec 27 12:42 johnbret
-rw-r--r--   1 reliable group     9728 Jan 01  1970 junk.msg
drwxr-xr-x   2 reliable group      128 Mar 10 09:08 lines
```

Following is a brief description of the commands of the Local menu.

List – Displays the contents of a directory. You will be prompted to choose either a detailed (long) or non-detailed (short) list and then to enter a filename; you may use a specific filename, a filename with wildcard characters (for example, “*”), or press ENTER to display all files in the current local directory.

Delete – Erases a single file or multiple files. You may use a specific filename or a filename with wildcard characters (for example, “*”).

Edit – Invokes the editor specified in the EDITOR environment variable (see EDITOR on page 10). vi is the default editor.

Rename – Renames a local file.

Type – Displays a local file in the scrolling area.

10.7x	**View –** Displays either a “snapshot” or a “movie” made using the Access menu (see “The Access Menu” on page 327).

Print – Prints a file to the local printer or print spooler as defined by the BPRINTER environment variable (see BPRINTER on page 9).

Chdir – Changes from the current local directory to one that you name. The current directory is displayed on the top line of the BLAST screen. BLAST will check this directory for any files that you specify with the Local menu commands.

System – Performs a local system command. At the prompt, type a system command and press ENTER. Alternatively, you may simply press ENTER and escape to a system prompt that takes over the BLAST display. Typing `EXIT` and pressing ENTER returns you to BLAST. When BLAST is started with the `-b` switch (or with the `-n` switch if the display has not been re-enabled through a script), you cannot escape to a system prompt (see "Command Line Switches" on page 10).

The Remote Menu

If you are using BLAST protocol, FTP, or Kermit protocol, the File-transfer menu contains a Remote command, which takes you to the Remote menu. The Remote menu allows a user with no knowledge of the remote operating system to manage files on that system.

FIGURE 4-5

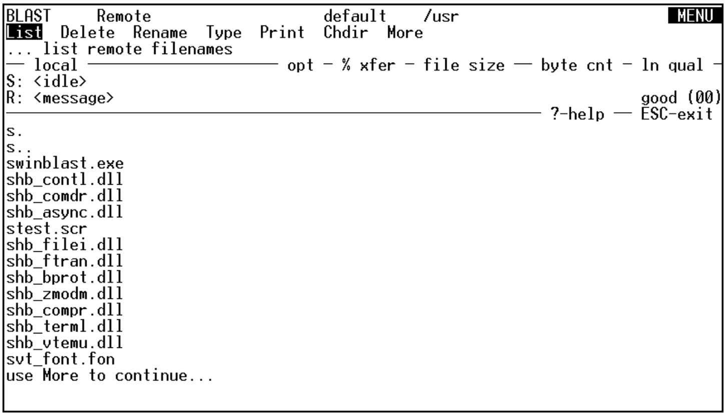


Figure 4-5 above shows the BLAST protocol Remote menu. The commands of this menu, which differ from the FTP and Kermit Remote menus, are described briefly below. For a fuller discussion of the commands of the Remote menus, see "BLAST Protocol Remote Menu" on page 118, "FTP Remote Menu" on page 128, and "Kermit Remote Menu" on page 134.

List –Lists a remote directory.

Delete – Deletes a single file or multiple files from the remote system.

Rename – Renames a remote file.

Type – Displays a remote file on the screen.

Print – Prints a remote file to the remote printer.

Chdir – Changes the current remote directory.

More – Scrolls a page of data output from the List or Type commands.

Automation with BLASTscript

Up to this point, you have been learning about BLAST in interactive mode, manually pressing keys to perform tasks. To automate communications tasks that are repeated on a daily or weekly basis, use BLAST's interpretive programming language, BLASTscript. BLAST scripts can:

◊ Automate the dial and logon sequences to another computer.

◊ Send and receive files.

◊ Control standard and nonstandard modems and communication devices.

◊ Customize the user interface.

◊ Perform error-checking for session validation.

◊ Access online information services to send and receive mail.

◊ Poll large numbers of unattended remote sites after regular business hours.

Refer to Chapters 12–14 and Appendix E of this manual for detailed information on the use of BLAST scripts.

Chapter 5

The Setup

What is a Setup?

Communication between computers requires a great deal of information: the phone number of the remote computer, the modem type and baud rate, basic communications parameters, and more. BLAST keeps this information in individual files called “setups,” one file for each different system connection. BLAST is distributed with blaster.su, a setup that contains the correct settings for you to call the BLAST demonstration line (see “Selecting the Blaster Setup” on page 45). A setup containing default values, default.su, is created when BLAST is executed for the first time.

You can customize the setup by selecting the Modify command in the Offline menu. Although this chapter tells you how to create, edit, and save setups, the Online Help for some setup fields has more specific information.

We recommend that you make any changes to the setup through the Modify menu; however, setups are text files and can thus be edited with any text editor. Be sure to save the file as “text only” or “ASCII” and give it the extension “.su”; do *not* save it as a word processor file.

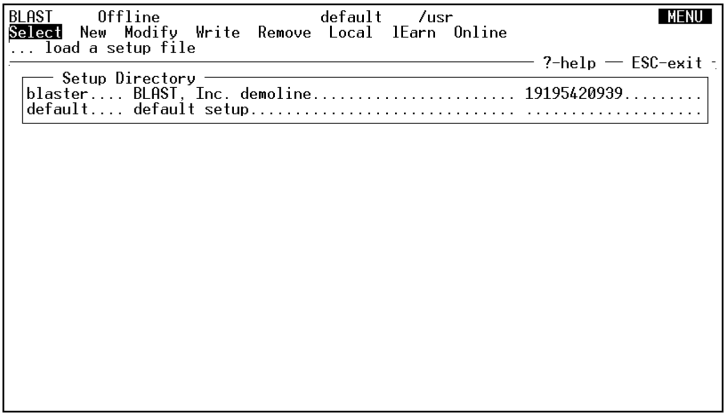
Loading a Setup

To load a setup, choose the Select command from the Offline menu (Figure 5-1). You will be prompted to enter a setup name or to press ENTER to see a Setup Directory of all available setup files. If you press ENTER to see the directory, use the keys listed at the top of the Command Area to highlight the setup that you want to load.

10.8x	Alternatively, you may use the cursor keys to highlight a setup file.

After highlighting a setup, press ENTER to load the selected setup.

FIGURE 5-1



The Default Setup

BLAST creates default.su, a setup that contains default values for each setup field and is automatically loaded when you start BLAST (unless you specify another setup on the command line). If you unintentionally overwrite the original default.su, you can restore its original settings by deleting or renaming the existing default.su and restarting BLAST. BLAST will create a new default.su.

Creating a New Setup

To create a new setup, select the New option from the Offline menu by pressing N. BLAST will prompt you for a new setup name. Note that BLAST may not display the entire filename in its Setup Directory. (Some UNIX systems have a limit of 256 characters for a filename; some UNIX systems have a limit of only 14 characters.) You may want to use the location of the remote site as the setup name, or some other easily remembered name. If you want to place the new setup in a subdirectory of the directory specified by the SETUPDIR environment variable, you must enter the relative path along with the setup name. BLAST will automatically append the extension ".su" to the filename. After you have typed in the setup name and pressed ENTER, BLAST will automatically enter Modify mode (see next section) and display in the Setup window the values of the setup file currently in memory. After you modify these values and press *CANCEL*, BLAST will automatically save the new setup file, load its values into memory, and return to the Offline menu.

Modifying a Setup

To modify a setup file from the Offline menu, use the Select command to load the setup into memory and then press M, for Modify. You will see a screen with a Setup window similar to the one shown in Figure 5-2 below:

FIGURE 5-2

```
BLAST     Offline                  blaster    /usr                    MENU
... ^E-up ^X-down ^R-first ^C-last ^T-clear <ESC>-exit
... press ^T to clear or enter new text
─────────────────────────────────────────────────────────── ?-help ── ESC-exit
 ┌─── Setup for: blaster ──────────────────────────────────────────────────┐
 │       Description: BLAST, Inc. demoline________________________         │
 │      Phone Number: 19195420939_____________________________             │
 │       System Type: UNIX____                                             │
 │            Userid: reliable________          Attention Key: ^K_         │
 │          Password: XXXXXXXXXXXXXXXX                                     │
 │        Connection: /dev/tty4A_______________     Emulation: VT100...    │
 │    Connection T/O: 60_                         Full Screen: YES         │
 │  Originate/Answer: ORIGINATE                    Local Echo: NO          │
 │        Modem Type: AT______                     AutoLF In: NO           │
 │         Baud Rate: 9600                        AutoLF Out: NO           │
 │            Parity: NONE                     Wait for Echo: NO           │
 │    Data/Stop Bits: 8/1                        Prompt Char: NONE         │
 │   XON/XOFF Pacing: YES                         Char Delay: 0__          │
 │   RTS/CTS Pacing: NO                           Line Delay: 0__          │
 │     Keyboard File: _______________________                              │
 │       Script File: _____________________________  Protocol: BLAST...    │
 │          Log File: _____________________________ Packet Size: 256_      │
 │   Translate File: _____________________________                         │
 └─────────────────────────────────────────────────────────────────────────┘
```

A field must be highlighted before you can modify its value. Use the keys listed at the top of the Command Area to move from field to field.

10.8x	You may also use the cursor keys to move from field to field.

The third line of the Command Area will indicate the type of action necessary to enter a value.

Most fields are multiple choice. Use the SPACEBAR to cycle forward (and the BACKSPACE to cycle backwards) through the available options in these fields; then press ENTER to proceed to the next field. Some fields, like Phone Number, require your input. To correct a mistake while entering data, use BACKSPACE to delete the mistake and then continue typing, or press CTRL T to clear the field and start over.

10.8x	Alternatively, you can press CTRL P to bring up the field for editing at the top of your screen. Edit using the BACKSPACE, DELETE, and cursor (arrow) keys; then press ENTER.

The Emulation and Protocol fields may require additional input. If the entry in the field is followed by three periods, it means that there is a subwindow of additional settings. Press ENTER to access a subwindow. After making the necessary changes to this subwindow, press the *CANCEL* key to return to the Modify menu.

The values to be used in this session are now stored in your system's memory and are known as the "current" setup. The program can continue without saving these changes to disk or you may save the altered setup for future use by using the Write command, which is highlighted when you exit Modify mode.

Removing a Setup

To delete a setup, choose Remove from the Offline menu. At the prompt, either

◊ type in the name of the setup you want to delete and press ENTER, or

◊ press ENTER to display the Setup Directory, highlight the setup you want to delete, and press ENTER.

You will then be asked if you want to delete the setup. Select "Yes" to delete the setup or "No" to cancel the deletion and return to the Offline menu.

Setup Fields

This section briefly discusses the function of each setup field of the Setup window and indicates default values in brackets and corresponding BLASTscript variables in italics (For more on BLASTscript variables, see Chapter 16). The Online Help for each field also contains detailed information. The individual fields are discussed on the pages listed in the following table:

Description — user-defined

Provides a detailed description of the setup. This is a free form comment; however, scripts can use the variable `@SYSDESC` for any purpose. For example, the program can take information from the description line as input or write to it to save status information.

BLASTscript variable: @SYSDESC

Phone Number — user-defined

Stores the phone number of the remote computer. This field will allow up to 40 characters. For a direct connection, leave the Phone Number field empty.

Although any alphanumeric characters may be entered, be careful to avoid using characters that may be misinterpreted by the modem. This string of characters is passed unchanged to the modem. See your modem manual for details.

BLASTscript variable: @PHONENO

System Type — any valid system type

Identifies the computer type to which BLAST will connect. If you are connecting to a system that does not appear in the System Type field or to a single-user system, select `NONE`. (Mac and PC types are provided for consistency with BLAST scripts but are equivalent to

NONE.) The CONNECT, DISCONNECT, FILETRANSFER, and UPLOAD processes use this information to automate your logons and file transfers.

The available system types are modified periodically by BLAST, Inc. The following example list may or may not include the system types available with your copy of BLAST. You may download the most recent system script from our FTP site at ftp://blast.com/dist/scripts/.

NONE – Single-user system such as IBM PC or Apple Macintosh
PC – IBM PC
Mac – Apple Macintosh
VMS – DEC VAX VMS
AOS – Data General AOS
BHost – BLAST Host
UNIX – UNIX
XENIX – Xenix
AIX – IBM RS/6000
A/UX – Apple UNIX
HP-UX – Hewlett-Packard UNIX
IRIX – Silicon Graphics UNIX
QNX – QNX 4.2
SCO – SCO UNIX
SunOS – Sun UNIX
Ultrix – DEC VAX Ultrix
CEO – Data General
MVS/TSO – IBM Mainframe
VM/CMS – IBM Mainframe
WBHOST – WinBLAST

To specify a user-defined system type, enter into this field the name of the .scr file. See Chapter 14 for more details on systems.scr and user-defined system scripts.

BLASTscript variable: @SYSTYPE

Userid — user-defined

Holds the login ID that you will use to log onto the remote system. With the value of this field, BLAST's CONNECT command uses the systems.scr library to answer logon queries automatically.

BLASTscript variable: @USERID

Password user-defined

Holds the password that you will use to log onto the remote system. With the value of this field, BLAST's `CONNECT` command uses the systems.scr library to answer password queries automatically. To maintain security, this field is intentionally overwritten with Xs in the Setup window and encoded in the setup file on the disk.

As additional security, BLAST will prompt you for this password if this field is left blank; therefore, the password need not be on the disk at all. See the explanation of `@PASSWORD` on page 280.

BLASTscript variable: @PASSWORD

Connection any valid device

Specifies the communications port, host system for TCP/IP connections, or hunt file that BLAST will use for the current session. Valid options are:

Device name – Any valid asynchronous port (e.g., /dev/tty1A).

Host name or address – The name or network address of the TCP/IP host system to which you want to connect (for example, "blaster.blast.com"). To establish a raw socket, enter the host name and any available port number except 23. Port number 23 is reserved for telnet. To use telnet, simply enter the host name, and BLAST will default to port number 23. To use telnet with a port other than port 23, enter the host name, the port number, and "telnet," as in the example below:

```
blaster.blast.com 12 telnet
```

See "Accessing TCP/IP Ports" on page 16.

Hunt filename – The name (including path) of a hunt file that lists available devices, preceded by the "<" character. Refer to "Automatic Serial Port Searching" on page 25 for details concerning hunt files.

BLASTscript variable: @COMMPORT

Connection T/O 0 – 999 **[60]**

For networks, specifies the number of seconds that BLAST will wait for a network connection after entering the Online menu. This field has no effect on serial connections.

If the specified amount of time passes and a connection has not been made, BLAST will display an error message, set @STATUS to a non-zero value, and return to the Online menu.

Values for this field range from 0 to 999 seconds. If set to 0, BLAST will not time out.

BLASTscript variable: @CONNTIMO

Originate/Answer **[ORIGINATE]** ANSWER

Specifies what BLAST will do during the automated connect and disconnect processes.

To dial out and initiate a connection, set the field to ORIGINATE. To set BLAST to wait for a caller to connect, set the field to ANSWER.

BLASTscript variable: @ORGANS

Modem Type any valid modem type

Identifies the modem connected to your communications port. When you select the Online Connect or Disconnect menu command, or use the CONNECT or DISCONNECT BLASTscript command, BLAST uses the modem type named in this field to execute pre-defined programs from the modems.scr library. These routines perform various hardware-specific tasks, such as dialing the phone and disconnecting from the remote computer.

The available modem types are modified periodically by BLAST, Inc. The following list may or may not include the modem types available with your copy of BLAST. You may download the most recent modems.scr from our FTP site at ftp://blast.com/dist/scripts/.

NONE – no modem specified
HARDWIRE – direct connection
APEX – Apex Data modems
AT – Generic AT command set (does not set flow control)
AT&T – AT&T Paradyne modems
BOCA – Boca modems
CARDINAL – Cardinal modems
CODEX – Codex modems
HAYES – Hayes modems
INTEL – Intel modems
MEGAHZ – Megaherz modems
MICROCOM – Microcom modems
MOTOROLA – Motorola Universal Data Systems (UDS) modems

`MULTITEC` – MultiTech modems
`OPTIMA` – Optima Hayes modems
`OSITECH` – Ositech modems
`PRACTICL` – Practical Peripherals modems
`SUPRA` – Supra modems
`TELEBIT` – Telebit modems
`UDSFASTK` – Motorola UDS FasTalk
`UDSV3229` – Motorola UDS V3229
`USROBOT` – U.S. Robotics modems
`USRV32` – U.S. Robotics Courier V.32, V.32bis, V.42, V.42bis
`ZOOM` – Zoom modems
`ZYXEL` – ZyXEL modems

If your modem does not appear as a choice in the setup field, you may specify a user-defined modem type by entering into this field the name of the .scr file. See Chapter 14 for more details on modems.scr and user-defined modem scripts.

BLASTscript variable: @MODEM

Baud Rate 300 600 1200 2400 4800 **[9600]** 19.2 38.4 57.6 115K

Specifies the speed at which the serial port device driver communicates with the modem. This may or may not be the same speed at which the modems communicate with each other. Some older modems are incapable of negotiating link speeds with other modems. A Hayes 2400, for example, will not operate at speeds any higher than 2400. If you have trouble connecting with other systems, match your Baud Rate setting with the highest Baud Rate supported by the remote system.

It is sometimes advantageous to run at a lower than maximum baud rate. If you have a slow computer, are running many applications simultaneously, or have limited system memory, you may notice dropped characters at very high baud rates, causing garbled displays in Terminal mode and a high number of block retransmissions during file transfers. Throughput may be better at a slower rate.

BLASTscript variable: @BAUDRATE

Parity **[NONE]** ODD EVEN

Sets the device driver parity of the serial port. This setting should match that of the remote system.

BLASTscript variable: @PARITY

Data/Stop Bits 7/1 7/2 **[8/1]** 8/2

Sets the number of data bits (7 or 8) and number of stop bits (1 or 2) for the device driver.

BLASTscript variable: @D/S_BITS

XON/XOFF Pacing YES **[NO]**

Specifies whether BLAST will use software flow control during text uploading, Terminal mode operation, and file transfer. When one computer needs to stop the flow of incoming data, it transmits an XOFF (CTRL S) to the other computer. When the computer is again ready to receive data, it transmits an XON (CTRL Q).

During BLAST protocol transfer, BLAST will wait a maximum of 30 seconds for an XON from the remote. If the XON is not sent, BLAST will resume transfer. See “Flow Control” on page 30.

BLASTscript variable: @XONXOFF

RTS/CTS Pacing YES **[NO]**

Enables hardware flow control. RTS/CTS pacing uses the RS-232 signals Request-to-Send and Clear-to-Send for optimized throughput over error-correcting modems. Not all systems support this type of flow control.

Set this field to `NO` unless error-correcting modems are on both ends of the connection. See “Flow Control” on page 30.

BLASTscript variable: @RTSCTS

10.7x

Keyboard File filename

Specifies a user-defined keyboard map for a particular keyboard or application (see “Keyboard Mapping Utility for 10.7x” on page 315).

BLASTscript variable: @KEYFILE

Script File filename

Designates a BLAST script that will be executed immediately when the setup is loaded into memory. A script specified on the BLAST command line will override a script specified in this field.

Use BLAST scripts to automate part or all of a BLAST session.

BLASTscript variable: @SCRFILE

Log File — filename

Names the log file that keeps a record of all session activity. When a file is transferred, a menu selection made, or a BLASTscript statement executed, the log file records the activity and the time that it occurred. Extended logging offers detailed information about file transfers. For more information on extended logging, see the description of the `@XLOG` reserved variable on page 298.

If the filename that you enter already exists, BLAST appends the new session activity information to the existing file; otherwise the file is created. Log files do not need any particular extension and can be any combination of the normally accepted filename characters. You may specify a full path as part of the log filename.

BLASTscript variable: @LOGFILE

Translate File — filename

Designates a control file to filter incoming or outgoing characters in Terminal mode and during text upload/capture. The Translate File is an ASCII text file that can be edited by a text processor or the BLAST editor. See "Translate File Format" on page 306 for more information.

BLASTscript variable: @XLTFILE

Attention Key — any Control key **[^K]**

Defines the key combination that will be interpreted as the Attention Key. This field accepts a single keystroke, which will be used in combination with the CTRL key. Throughout this manual, the Attention Key is referred to as *ATTN*.

If it is necessary to change the attention key, be sure to choose a replacement value that will not interfere with your system's designated control codes. In particular, do not use `^M`, which is the control code for a carriage return. Check your system manual for more information about special control codes *before* you reassign the attention key.

10.8x	You can turn off the attention key in a script by setting @ATTKEY to a null value (""). When the script terminates, its value is reset to its previous setting.

We recommend that you do not change this setting.

BLASTscript variable: @ATTKEY

Emulation

10.7x	PASSTHRU **[VT320]** any valid terminal emulator The terminal values are PASSTHRU, in which the characters received by the serial port are displayed without change, TTY, and the following terminal emulators: VT320, VT220, VT100, VT52, PC ANSI, TV920, D80, ADM3A, WYSE50, and WYSE60.
10.8x	TTY **[PASSTHRU]** The terminal values are PASSTHRU, in which the characters received by the serial port are displayed without change, and TTY.

BLASTscript variable: @EMULATE

DEC VT Emulation Subwindow for 10.7x

Selecting any of the VT emulators and pressing ENTER will display a subwindow of extended configuration options. The VT320 subwindow is pictured in Figure 5-3; the VT220, VT100, and VT52 subwindows are variations of the VT320 subwindow. Not all of the following setup fields will appear in every subwindow.

FIGURE 5-3

```
VT320 Emulation

  7/8 Bit Controls: 7                    User Def Keys: UNLOCKED
    80/132 Columns: 80                     Text Cursor: YES

 Horizontal Scroll: JUMP                     Auto Wrap: NO
   Jump Scroll Inc: 10                        New Line: NO
       Keypad Mode: NUMERIC                 Print Mode: NORMAL
  Cursor Keys Mode: NORMAL                Print Screen: SCROLL REGION
                                         Intl Char Set: USASCII
    Reset Terminal: NO               User Pref Char Set: DEC SUPPLEMENTAL
      Clear Screen: NO

    Answerback Msg: ______________________________
```

7/8 Bit Controls **[7]** 8

Specifies whether "CI" control characters are represented in the 8-bit environment or as 7-bit escape sequences.

BLASTscript variable: @VT8BIT

80/132 Columns **[80]** 132

Toggles between 80-column and 132-column display for text.

BLASTscript variable: @VTDISP132

Horizontal Scroll **[JUMP]** NONE SMOOTH

Specifies how to scroll data on an 80-column display when the emulator is in 132-column mode. `SMOOTH` scroll will move the view of the display only as necessary to display the cursor position. `JUMP` scroll will adjust the view by showing either the first 80 columns or the last 80 columns. When `NONE` is selected, the display will not scroll and the cursor may disappear from view.

BLASTscript variable: @VTHSCROLL

Jump Scroll Inc 1 – 53 **[10]**

Specifies the number of columns to scroll when the Scroll Left or Scroll Right keys are pressed and `Jump` has been selected as the setting for Horizontal Scroll. Acceptable values are `1–53`.

BLASTscript variable: @VTHSCROLLN

Keypad Mode **[NUMERIC]** APPLICATION

Specifies whether the numeric keypad keys will send numbers (`NUMERIC`) or programming functions (`APPLICATION`) defined by the application.

BLASTscript variable: @VTKEYPAD

Cursor Keys Mode **[NORMAL]** APPLICATION

Specifies whether the cursor keys will control cursor movement (`NORMAL`) or send application control functions (`APPLICATION`).

BLASTscript variable: @VTCURSOR

Reset Terminal — YES **[NO]**

Specifies resetting many of the VT320 operating features, such as scrolling regions and character attributes, to their factory default values upon entering Terminal mode. YES resets these values; the value of this variable is then automatically reset to NO.

BLASTscript variable: @VTRESET

Clear Screen — YES **[NO]**

Specifies clearing of the terminal's video display the next time you enter Terminal mode. YES clears the terminal's video display; the value of this variable is then automatically reset to NO.

BLASTscript variable: @VTCLRSCRN

Answerback Msg — up to 30 characters

Contains a message to be sent to the remote computer upon receiving an inquiry (^ E). The message can be up to 30 characters in length.

BLASTscript variable: @VTANSBACK

User Def Keys — **[UNLOCKED]** LOCKED

Specifies whether the host system can change user-defined key (UDK) definitions.

BLASTscript variable: @VTUSERKEYS

Text Cursor — **[YES]** NO

Specifies whether to display the text cursor.

BLASTscript variable: @VTTEXTCURS

Cursor Type — BLOCK **[LINE]**

Specifies whether the cursor is displayed as a reverse-video block or as an underline character.

BLASTscript variable: @VTCURSTYPE

Auto Wrap YES **[NO]**

Specifies whether text typed at the right margin will automatically wrap to the next line.

BLASTscript variable: @VTAUTOWRAP

New Line YES **[NO]**

Selects whether the ENTER key will move the cursor to a new line. Possible choices are `NO` (the ENTER key sends only a carriage return) and `YES` (both a carriage return and line feed are sent).

BLASTscript variable: @VTNEWLINE

Print Mode **[NORMAL]** AUTO CONTROLLER

Specifies when information is sent to the printer. In `AUTO` print mode, each line of received text is displayed and printed; in `CONTROLLER` mode, all received data is sent directly to the printer without displaying it on the screen; and in `NORMAL` mode, the user initiates printing from the keyboard.

BLASTscript variable: @VTPRINT

Print Screen **[SCROLL REGION]** FULL PAGE

Specifies how much of the screen to print when you press the PRINT SCREEN key. Choices are `FULL PAGE` (entire page) and `SCROLL REGION` (only the currently defined VT scrolling region).

BLASTscript variable: @VTPRINTPAGE

Intl Char Set **[USASCII]** UK FRENCH GERMAN ITALIAN SPANISH DANISH

Specifies whether 7- or 8-bit data is used for international support. The default value is `USASCII`, which allows 8-bit data. The high-order values are used to represent international characters. If any other character set is selected, specific international characters replace characters within the ASCII set.

BLASTscript variable: @VTINTL

User Pref Char Set — **[DEC SUPPLEMENTAL]** ISO LATIN-1

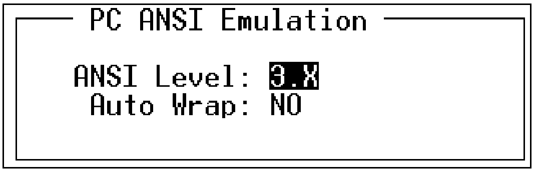
Selects either `DEC SUPPLEMENTAL` or `ISO LATIN-1` as the user preferred character set.

BLASTscript variable: @VTUSERCHAR

PC ANSI Emulation Subwindow for 10.7x

Selecting the PC ANSI emulator and pressing ENTER displays a subwindow of extended configuration options shown in Figure 5-4:

FIGURE 5-4



ANSI Level — 2.x **[3.x]**

Specifies the level of ANSI for your system. Some applications require ANSI Level `2.x`.

BLASTscript variable: @ANSILEVEL

Auto Wrap — YES **[NO]**

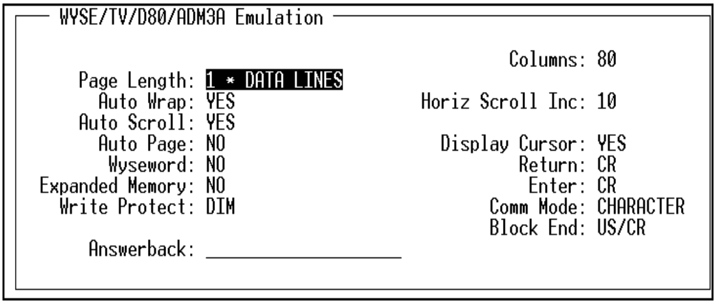
Specifies automatic wrapping of lines longer than 80 characters.

BLASTscript variable: @ANSIAUTOWRAP

WYSE Emulation Subwindow for 10.7x

Selecting the WYSE60, WYSE50, TV920, D80, or ADM3A emulators and pressing ENTER will display a subwindow of extended configuration options shown in Figure 5-5 (next page).

FIGURE 5-5



Page Length **[1 * DATA LINES]** 2 * DATA LINES 4 * DATA LINES

Sets page length in number of screens of data. `2* DATA LINES` sets the page length to 48 lines; `4* DATA LINES` sets the page length to 96 lines.

BLASTscript variable: @WYPAGELEN

Auto Wrap **[YES]** NO

Specifies whether a new line is automatically performed when a character is placed in the last column of a row (column 80 or 132).

BLASTscript variable: @WYAUTOWRAP

Auto Scroll **[YES]**

Specifies scrolling of the terminal display when the cursor reaches the bottom of a page. The default value of YES cannot be changed. (The Auto Scroll value is ignored if Auto Page is on.)

BLASTscript variable: @WYAUTOSCROLL

Auto Page YES **[NO]**

Controls whether the cursor can move off the current page. If YES is selected, the cursor can move above the first line to the previous page or below the last line to the next page.

BLASTscript variable: @WYAUTOPAGE

Wyseword YES **[NO]**

Specifies whether keys send Wordstar™ functions (YES) or the standard key codes (NO). The only keys that are affected are the WYSE keys that can be mapped with the blastkbd utility (see “Keyboard Mapping Utility for 10.7x” on page 315).

BLASTscript variable: @WYSEWORD

Expanded Memory YES **[NO]**

Toggles “expanded” memory use. Note that this is *not* related to DOS “expanded memory.” Normally, the terminal emulator uses two pages of video display memory. If the maximum of four pages are required, expanded memory must be set to YES. Note, however, that more run-time memory will be required by BLAST, possibly adversely affecting throughput during file transfers.

BLASTscript variable: @WYEXPNDMEM

Write Protect **[DIM]** REVERSE NORMAL

Specifies the attributes used to display protected fields.

BLASTscript variable: @WYWRITEPROT

Answerback up to 20 characters

Contains a message to be sent to the remote computer upon receiving an inquiry (^E). The message can be up to 20 characters long.

BLASTscript variable: @WYANSBACK

Columns **[80]** 132

Specifies 80 or 132 columns per row.

BLASTscript variable: @WYDISP80

Horiz Scroll Inc 1 – 53 **[10]**

Specifies the number of columns to scroll when the cursor reaches a column that is not currently displayed. This value is used when 132 columns per row has been selected and compressed display is not available. Note that a value of 1 implies smooth scrolling. Any other value implies jump scrolling.

BLASTscript variable: @WYSCROLLINC

Display Cursor [YES]

Specifies that the cursor is visible. The default of YES cannot be changed.

BLASTscript variable: @WYDSPCURSOR

Return **[CR]** CRLF TAB

Selects the character to send when the RETURN key is pressed. The default value is CR, which signifies a carriage return.

BLASTscript variable: @WYRETURN

Enter **[CR]** CRLF TAB

Selects the character to send when the keypad ENTER key is pressed.

BLASTscript variable: @WYENTER

Comm Mode **[CHARACTER]** BLOCK

Controls whether data is sent after each keystroke (CHARACTER mode) or grouped in blocks (BLOCK mode).

BLASTscript variable: @WYCOMMODE

Block End **[US/CR]** CRLF/ETX

Specifies what characters mark the end-of-line and end-of-block when the terminal is in block mode and sends a block of data. If US/CR is selected, a US character (\037) is sent at the end of each line and a CR character (\015) is sent to mark the end of the block.

BLASTscript variable: @WYBLOCKEND

END OF EMULATION SUBWINDOW DESCRIPTIONS

Full Screen **[YES]** NO

Indicates whether the top four lines of the menu display will be suppressed while in Terminal mode. The default value is YES, which suppresses the menu and allows the top 24 lines of the terminal screen to be used for data.

BLASTscript variable: @FULLSCR

Local Echo YES **[NO]**

Specifies whether BLAST will echo typed characters to the screen while in Terminal mode. If this field is set to YES, BLAST will display typed characters before sending them out the communication port; if the field is set to NO, the characters will be displayed only if the remote computer sends them back.

If this field is set to YES and double characters are displayed on the screen, change the setting to NO.

BLASTscript variable: @LOCECHO

AutoLF In YES **[NO]**

Controls the Terminal mode actions when receiving carriage returns. Some remote systems do not automatically supply line feeds, causing multiple lines of text written on top of each other on your monitor. Set to YES to read incoming text correctly from this computer type. The setting for AutoLF In has no effect on text received in Capture mode.

BLASTscript variable: @AUTOLFIN

AutoLF Out YES **[NO]**

Controls Terminal mode actions when sending carriage returns. A setting of YES causes BLAST to append a line feed to each carriage return sent out from the communications port. Line feeds are often stripped from the data stream to increase throughput. Set this to YES if the remote system requires a line feed after the carriage return.

BLASTscript variable: @AUTOLFOUT

Wait for Echo YES **[NO]**

During text uploads, forces BLAST to wait for the echo of the previously sent character before sending another character; the setting has no effect on file transfers.

Wait for Echo "paces" text uploads to slow BLAST down when the remote computer operates more slowly than the local system. It is also useful when sending one-line commands to modems that cannot take bursts of high speed data while in Command mode.

BLASTscript variable: @WT4ECHO

Prompt Char **[NONE]** any ASCII character

Defines the character that BLAST will use to determine when to resume sending text. After sending a line of text and a carriage return, BLAST pauses until the remote system sends the prompt character. Prompting is an effective form of flow control while uploading text.

Any single character, including a control character, is a valid entry. To enter a control character, prefix the character with a caret (^). `NONE` disables prompting.

BLASTscript variable: @PROMPTCH

Char Delay **[0]** – 999

Specifies the time period (in hundredths of a second) that BLAST pauses between sending characters to the remote computer. This pause slows down strings sent by BLAST scripts and text that is uploaded.

Character delay is a form of flow control. Use this field when the remote computer is unable to keep pace with BLAST and no other form of flow control is available or to slow down the interaction with a modem or other simple hardware device that does not support other forms of flow control. The default value, `0`, specifies no delay. Character delay applies only to text uploads; it has no effect on file transfers.

BLASTscript variable: @CHARDLY

Line Delay **[0]** – 999

Specifies the length of time (in tenths of a second) to pause after sending a line of data. Line Delay provides a form of flow control while uploading text to the remote computer. Some remote systems may be unable to keep pace with BLAST; setting this field to a nonzero value can prevent overloading the remote computer. If `0` is entered, no delay will occur. Note that the setting for Line Delay applies only to text uploads.

BLASTscript variable: LINEDLY

Protocol — [BLAST] KERMIT XMODEM XMODEM1K YMODEM YMODEM G ZMODEM

10.8x	FTP

Selects the protocol that will be used for file transfers. The BLAST protocol generally runs faster and offers more features than other protocols.

BLASTscript variable: @PROTOCOL

BLAST Protocol Subwindow

Selecting BLAST and pressing ENTER displays the subwindow shown below in Figure 5-6:

FIGURE 5-6

```
┌── BLAST Protocol ──────────────────────────────────────────────────────────┐
│                                                                             │
│        Logon T/O: 120            ACK Request Frequency: 4_                  │
│   Inactivity T/O: 120       Number of Disconnect Blocks: 3                  │
│    7 Bit Channel: NO                      Launch String: \r________         │
│      Window Size: 16                  Transfer Password: ________           │
│DCD Loss Response: IGNORE            Enable /FWD and /STR: NO                │
│ Use "A" Protocol: NO         Enable /OVW and Remote Cmds: YES               │
│        Filtering: OFF           Send Compression Level: 4                   │
│ Retransmit timer: 4___        Receive Compression Level: 4                  │
│                                                                             │
└─────────────────────────────────────────────────────────────────────────────┘
```

Logon T/O — 0 – 999 **[120]**

Specifies the number of seconds that BLAST will attempt to establish a filetransfer session with the remote computer. Logon Timeout affects BLAST protocol transfers and remote control sessions. Timeouts can happen if:

- ◊ There is excessive noise on the line.
- ◊ There are parity or data/stop bit mismatches.
- ◊ BLAST is terminated unexpectedly on the remote computer.
- ◊ The connection is lost.

If zero is entered, no timeout will occur and BLAST will attempt to establish a filetransfer session with the remote computer indefinitely.

BLASTscript variable: @LOGTIMO

Inactivity T/O 0 – 999 **[120]**

Defines the time interval (in seconds) that BLAST will stay connected after the last valid data packet has been received from the remote computer. Timeouts happen if:

- ◊ The connection is lost.
- ◊ There is excessive noise on the line.
- ◊ The remote computer goes down.
- ◊ Flow control has not been released.

If zero is specified, BLAST never times out.

NOTE: In previous versions of BLAST, this field was named "Connect Timeout" and was associated with the BLASTscript reserved variable `@CONTIMO`.

BLASTscript variable: @INACTIMO

7-Bit Channel YES **[NO]**

Defines the logical width of the data path to be used. `YES` specifies a 7-bit data encoding scheme; `NO` specifies an 8-bit encoding scheme.

Some networks, minicomputers, and asynchronous devices only support 7-bit path widths. The BLAST protocol operates more efficiently using 8-bit encoding; however, the data path width has nothing to do with the type of data that can be transferred. BLAST may transfer 8-bit binary or 7-bit ASCII over either 7- or 8-bit data paths.

BLASTscript variable: @7BITCHN

Window Size 1 – **[16]**

Specifies the number of packets that can be sent to the remote without BLAST waiting for an acknowledgement from the remote. As packets are acknowledged, the starting point of the window adjusts, or "slides." For example, if the window size is `12` and the first 6 of 8 packets sent have been acknowledged, the start point of the window moves by 6, and 10 additional packets can be sent before BLAST must stop and wait for an acknowledgement. See "BLAST Protocol Design" on page 101 for a fuller discussion of window size.

BLASTscript variable: @WDWSIZ

DCD Loss Response — ABORT **[IGNORE]**

Specifies the action BLAST will take after DCD loss during a file-transfer session:

`ABORT` –Sets `@EFERROR` on carrier loss and exits Filetransfer mode.

`IGNORE` –Ignores carrier loss. Filetransfer mode continues until the Inactivity T/O takes effect.

BLASTscript variable: @DCDLOSS

Use "A" Protocol — YES **[NO]**

Specifies whether the BLAST "A" Protocol will be used. `YES` specifies communication with older BLAST products.

BLASTscript variable: @APROTO

Filtering — ON **[OFF]**

Specifies filtering out VT sequences sent from a remote computer or protocol converter. This filtering prevents BLAST protocol from labeling these sequences as bad blocks received.

BLASTscript variable: @FILTER

Retransmit Timer — 0 – 9999 **[4]**

Sets the maximum number of seconds BLAST will pause before resending a packet. For example, if Window Size is set to `5` and Retransmit Timer is set to `30`, BLAST will attempt to resend the fifth packet every thirty seconds if it receives no acknowledgement.

NOTE: This setting should be less than the that for Inactivity Time-out.

BLASTscript variable: @RETRAN

ACK Request Frequency — 1 – window size **[4]**

Specifies the frequency at which an acknowledgement from the receiving system is requested. The frequency is measured in number of packets sent. For example, if the ACK Request Frequency is `4`, a request for an acknowledgement is sent to the receiving computer every four packets. Set this field higher for better performance with error-correcting modems. See also Window Size setup field (page 85).

BLASTscript variable: @ACKFREQ

Number of Disconnect Blocks 0 – 9 **[3]**

Sets the number of *additional* disconnect blocks (after the first disconnect block) that BLAST sends when exiting Filetransfer mode. The default value is `3`, which indicates four total disconnect blocks.

BLASTscript variable: @NUMDISC

Launch String any ASCII string **[\r]**

Specifies a string to be appended to BLAST protocol blocks. This will help communications to a mainframe through protocol converters. Just as in BLASTscript, you may send any string of ASCII characters, including the same control characters used in string constants. Nonprintable characters can be represented with a back-slash followed by a three-digit octal number (for example, a line-feed may be represented as a \012). The string should not be enclosed in quotes. The default for this field is a carriage return (`\r`).

BLASTscript variable: @LAUNCHST

Transfer Password user-defined

Stores a case-sensitive password (up to eight characters) that restricts a remote user's access. Requests to get files from a password-protected computer and to do file maintenance functions are not honored unless the password is received first. Without the password, the remote machine is limited to sending and receiving messages.

To send the Transfer Password, the remote user should select the Send menu command from the Filetransfer menu; then, at the local filename prompt, type the following:

```
!password=your_password
```

where *`your_password`* is the transfer password. The remote filename field and transfer options should be left blank. In a BLAST script, the `SEND` statement should be followed by a line with the password and then two blank lines (See "Using the Transfer Password" on page 121).

The transfer password is superseded by the Secure BLAST password described in Chapter 11. See that chapter for further details.

NOTE: The Transfer Password is intended to validate remote users logging onto your system. If a local operator uses a setup with a Transfer Password entered, he or she will not be able to receive files without the remote computer sending the password.

BLASTscript variable: @TRPASSWD

Enable /FWD and /STR YES **[NO]**

Enables the /FWD and /STR file transfer switches. Note that disabling these switches affects only *local* files. For example, you will still be able to get a file with the /FWD switch, because the successfully transferred file will be deleted from the *remote* system.

BLASTscript variable: @ENABLEFS

Enable /OVW and Remote Cmds **[YES]** NO

Enables the /OVW file transfer switch and system commands received during BLAST Protocol Filetransfer mode. Disabling /OVW affects only *local* files. For example, you will still be able to send a file with the /OVW switch because the file will be overwritten on the *remote* system. The List, Type, and More commands remain active when this field is set to `NO`; only potentially destructive commands are disabled.

BLASTscript variable: @ENABLERCMD

Send Compression Level 0 – 6 **[4]**

Specifies the maximum compression level to be used while sending files to the remote computer. Level `0` specifies no compression; level `6` specifies the highest compression level.

BLASTscript variable: @SCOMP_LEV

Receive Compression Level 0 – 6 **[4]**

Specifies the maximum compression level to be used while receiving files from the remote computer. Level `0` specifies no compression; level `6` specifies the highest compression level.

BLASTscript variable: @RCOMP_LEV

Kermit Protocol Subwindow

Selecting `KERMIT` and pressing ENTER displays the subwindow shown in Figure 5-7 below:

FIGURE 5-7

```
KERMIT Protocol

        Send Parameters                          Receive Parameters

  Start-of-Packet Char: ^A                  Start-of-Packet Char: ^A_
    End-of-Packet Char: ^M_                   End-of-Packet Char: ^M_
 Packet Size (maximum): 90__               Packet Size (maximum): 90__
        Pad Character: ^@_                        Pad Character: ^@_
              Padding: 0_                               Padding: 0_
                                                Timeout (seconds): 10
        Transfer Type: BINARY                 Filename Conversion: YES
                                                  Incomplete File: DISCARD
                Delay: 5_                                 Warning: ON
      Block-Check-Type: 2
```

Start-of-Packet Char **[^A]** – ^Z

For sending files with Kermit: specifies a control character to precede each packet sent from the local computer. The same control character must also be used by the remote Kermit.

BLASTscript variable: @KSSOPKT

For receiving files with Kermit: specifies a control character to precede each packet received by the local computer. The same control character must also be used by the remote Kermit.

BLASTscript variable: @KRSOPKT

End-of-Packet Char ^A – ^Z **[^M]**

For sending files with Kermit: specifies a control character to terminate each packet sent from the local computer. The same control character must also be used by the remote Kermit.

BLASTscript variable: @KSEOPKT

For receiving files with Kermit: specifies a control character to terminate each packet received by the local computer. The same control character must also be used by the remote Kermit.

BLASTscript variable: @KREOPKT

Packet Size 10 – 2000 **[90]**

For sending files with Kermit: specifies the packet size that your system will use when it transmits a file. Note that the remote Kermit server's Receive Packet Size should also be set to this value. The larger the packet, the more efficient the transfer; however, larger packets will pose problems on a noisy connection. Set larger packet sizes when there is little line noise, you are communicating with a mainframe, or you are using V.29 "ping pong" modems.

BLASTscript variable: @KSPKTLEN

For receiving files with Kermit: specifies the packet size that your system will use when it receives a file. Note that the remote Kermit server's Send Packet Size should also be set to this value. The larger the packet, the more efficient the transfer; however, larger packets will pose problems on a noisy connection. Set larger packet sizes when there is little line noise, you are communicating with a mainframe, or you are using V.29 "ping pong" modems.

BLASTscript variable: @KRPKTLEN

Pad Character **[^@]**, ^A – ^Z

For sending files with Kermit: specifies an alternate character to pad each packet transmitted by the local computer.

BLASTscript variable: @KSPADCH

For receiving files with Kermit: specifies an alternate character to pad each packet received by the local computer

BLASTscript variable: @KRPADCH

Padding **[0]** – 99

For sending files with Kermit: specifies the number of padding characters to send per packet. Padding can induce delays during a Kermit file transfer, allowing slower machines or older versions of Kermit more time to process the data you send.

BLASTscript variable: @KSPADDNG

For receiving files with Kermit: specifies the number of padding characters to request per packet. Padding can induce delays during a

Kermit file transfer, allowing slower machines or older versions of Kermit more time to process the data you receive.

BLASTscript variable: @KRPADDNG

Transfer Type TEXT **[BINARY]**

Specifies the type of file being transferred. Text files will be converted to local format.

BLASTscript variable: @KFILETYP

Delay 1 – 99 **[5]**

Specifies the number of seconds of delay between the recognition of a Send command and the actual beginning of the transmission.

BLASTscript variable: @KDELAYOS

Block-Check-Type 1 – 3 **[2]**

Specifies level of error detection. Kermit offers three levels of error detection, with 3 being the most secure. To decrease the chance of a bad packet being accepted by the receiving computer, set the level to 2 or 3. Higher levels of error detection will appreciably slow a file transfer. Use a lower block-check-type when using error-correcting modems or when transferring files at 9600 baud and above.

BLASTscript variable: @KBCHECK

Timeout 0 – 99 **[10]**

Specifies the number of seconds that the computer will wait to receive a packet before requesting that it be resent.

BLASTscript variable: @KRTIMEOUT

Filename Conversion **[YES]** NO

Specifies whether to convert a filename from local format to common Kermit format. For example, lower case is changed to all uppercase; and “~”, “#”, and all periods after the initial one are converted to “x”s.

BLASTscript variable: @KFNAMCONV

Incomplete File [DISCARD] KEEP

Specifies whether to KEEP or DISCARD files incompletely received, such as a file being transferred when you abort a Get command. This insures that any file received is complete.

BLASTscript variable: @KSAVEINC

Warning [ON] OFF

For Kermit transfers, specifies whether Kermit will automatically rename a received file if another file with the same name already exists in the current directory. If the field is set to ON, Kermit will rename the file, adding a number (0001, 0002, etc.); if the field set to OFF, Kermit overwrites the file.

BLASTscript variable: @KWARNING

Xmodem and Ymodem Protocol Subwindow for 10.8x

In BLAST Professional UNIX 10.8x, selecting XMODEM, XMODEM1K, YMODEM, or YMODEM G and pressing ENTER displays the subwindow shown in Figure 5-8. *Some fields apply to Xmodem only*.

FIGURE 5-8

```
BLAST     Offline                  default    /u/matt1/work             MENU
... ^E-up ^X-down ^R-first ^C-last ^P-prompt ^T-clear ^A-exit
... enter a new number

      XMODEM and YMODEM Protocol

            Send Parameters                       Receive Parameters

              EOT Timeout: 100_
            Pad Character: 0__
          File Conversion: BINARY             File Conversion: BINARY
  Remote Line Termination: CR/LF      Remote Line Termination: CR/LF
          Error Detection: CRC
```

EOT Timeout 10 – 6000 [100]

For Xmodem and Ymodem transfers, specifies EOT (end-of-transmission) timeout in hundredths of a second.

BLASTscript variable: @XYEOT

Pad Character any character in decimal **[00]**

For Xmodem transfers, specifies the pad character.

BLASTscript variable: @XPADC

File Conversion ASCII **[BINARY]**

For sending Xmodem and Ymodem transfers, specifies conversion to ASCII.

BLASTscript variable: @XYCONVS

For receiving Xmodem and Ymodem transfers, specifies conversion to ASCII.

BLASTscript variable: @XYCONVR

Remote Line Termination CR **[CR/LF]**

For sending files with Xmodem or Ymodem: specifies how line termination is treated.

`CR` – For files sent, replaces line feeds (`LF`) with carriage returns (CR); for example, when ASCII files are sent to a Macintosh platform.

`CR/LF` – For files sent, adds a carriage return (`CR`) before a line feed (`LF`); for example, when ASCII files are sent to DOS or Windows platforms.

BLASTscript variable: @XYRLTS

For receiving Xmodem and Ymodem files: specifies how line termination is treated.

`CR` – For files received, replaces all carriage returns (`CR`) with linefeeds (`LF`); for example, when ASCII files are received from a Macintosh platform.

`CR/LF` – For files received, deletes any carriage return (`CR`) *that is followed by a line feed* (`LF`); for example, when ASCII files are sent to DOS or Windows platforms.

BLASTscript variable: @XYRLTR

Error Detection **[CRC]** CHECKSUM

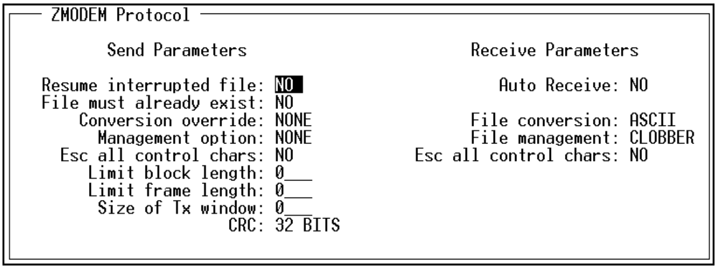
For Xmodem transfers, specifies whether the error detection is `CRC` or `CHECKSUM`.

BLASTscript variable: @XCRC

Zmodem Protocol Subwindow

Selecting `ZMODEM` and pressing ENTER displays the subwindow shown in Figure 5-9 below.

FIGURE 5-9



Resume Interrupted File YES **[NO]**

Continues an aborted binary file transfer from the point of interruption. The destination file must already exist and be smaller than the source file.

BLASTscript variable: @ZMRESUME

File Must Already Exist YES **[NO]**

Transfers the file only if it already exists on the destination system.

BLASTscript variable: @ZMEXIST

Conversion Override **[NONE]** ASCII BINARY

Allows the sender to specify to the receiver whether the data should be treated as `BINARY` or `ASCII` data, overriding the File Conversion setting of the receiving system. If `NONE` is selected, the data is handled according to the receiver's file conversion parameter.

BLASTscript variable: @ZMCONVS

10.8x	**ASCII Line Termination** **[CR/LF]** CR For sending ASCII files to nonstandard implementations of Zmodem, specifies line-feed conversion for ASCII files. When `@ZMCONVS = "ASCII"`, the default `CR/LF` specifies that line feeds be converted to `CR/LF`; `CR` specifies no conversion. *BLASTscript variable: @ZMALT*

Management Option **[NONE]** PROTECT CLOBBER NEWER NEWER/LONGER DIFFERENT APPEND

Specifies a file management option for files sent. Possible values are:

`NONE` – The file is transferred if it does not already exist on the receiving system.

`PROTECT` – The file is transferred only if it does not already exist on the receiving system, even if the receiving system has specified `CLOBBER`

`CLOBBER` – The file is transferred whether or not it already exists on the receiving system, unless the receiving system has specified `PROTECT`.

`NEWER` –The file is transferred if it does not already exist on the receiving system, or if the source file is newer (by date).

`NEWER/LONGER` –The file is transferred if it does not already exist on the receiving system, or if the source file is newer (by date) or longer (in bytes).

`DIFFERENT` – The file is transferred if it does not already exist on the receiving system, or if the files have different lengths or dates.

`APPEND` –The file is appended to a file of the same name on the receiving system based on the value of the receiving system's "File conversion" setting.

BLASTscript variable: @ZMMANAGS

Esc All Control Chars — YES **[NO]**

For sending files with Zmodem: specifies that all control characters sent will be link-escape encoded for transparency. By default, only the characters represented by hexadecimal 10, 11, 13, 90, 91, and 93, and the sequence "@-CR" are link-escape encoded.

BLASTscript variable: @ZMCTLESCS

For receiving files with Zmodem: specifies that all control characters received will be link-escape encoded for transparency. By default, only the characters represented by hexadecimal 10, 11, 13, 90, 91, and 93, and the sequence "@-CR" are link-escape encoded.

BLASTscript variable: @ZMCTLESCR

Limit Block Length — **[0]** 24 – 1024

Overrides the default block length, which is determined by the Baud Rate of the connection.

Baud Rate	Block Length (in bytes)
`300`	`128`
`600, 1200`	`256`
`2400`	`512`
`4800` or greater	`1024`

Specifying a value between `24` and `1024` limits the block length to the new value. A value of `0` specifies the default block length as determined by the baud rate.

BLASTscript variable: @ZMBLKLN

Limit Frame Length — **[0]** 24 – 1024

For Zmodem transfers, limits frame length and forces the sender to wait for a response from the receiver before sending the next frame. The default, `0`, specifies no limit to frame length.

BLASTscript variable: @ZMFRMLEN

Size of Tx Window — **[0]** – 9999

Specifies the size of the transmit window, which regulates how many data subpackets can be "outstanding" (unacknowledged) before the sender quits sending and waits for acknowledgements. A value of `0` specifies no limit to window size.

BLASTscript variable: @ZMWINDOW

CRC 16 **[32]**

Specifies the CRC error-detection method to be used, either 16-bit or 32-bit.

BLASTscript variable: @ZMCRC

Auto Receive YES **[NO]**

Specifies Auto Receive mode, which begins downloading immediately after entering Filetransfer mode.

BLASTscript variable: @ZMAUTODOWN

File Conversion **[ASCII]** BINARY

Specifies whether received files will be treated as `ASCII` or `BINARY`. For correct file conversion to ASCII, the remote computer must send the files as ASCII.

BLASTscript variable: @ZMCONVR

File Management NONE PROTECT **[CLOBBER]** APPEND

Specifies a file management option for files received. Possible values are:

`NONE` – The file is transferred if it does not already exist on the receiving system.

`PROTECT` – The file is transferred only if it does not already exist on the receiving system, even if the sending system has specified `CLOBBER`.

`CLOBBER` – The file is transferred whether or not it already exists on the receiving system, unless the sending system has specified `PROTECT`.

`APPEND` – The file is appended to a file of the same name on the receiving system based on the value of the receiving system's "File conversion" setting.

BLASTscript variable: @ZMMANAGR

END OF PROTOCOL SUBWINDOW DESCRIPTIONS

Packet Size 1 – 4085 **[256]**

For BLAST protocol transfers, specifies the packet size that your system will use when it transfers a file. The larger the packet, the more efficient the transfer; however, larger packets will pose problems on a noisy connection. Use larger packet sizes when there is little line noise, you are communicating with a mainframe, or you are using V.29 “ping pong” modems.

This field “negotiates” down. The versions of BLAST running on the local computer and the remote computer will compare values and use the smaller of the two values.

While transferring files, watch the line quality and retry count in the upper right part of the screen. If the quality of the line varies, or there are a significant number of retries (more than one retry in 20–50 blocks), a smaller packet size will usually improve throughput. The default for this field is `256`, which is the optimum setting for most users.

IMPORTANT: When transferring files with BHOST, always set the Packet Size to at least `200`, which is BHOST’s minimum packet size.

BLASTscript variable: @PAKTSZ

Chapter 6

BLAST Session Protocol

What is a Protocol?

In the serial communications world, a "protocol" is a set of rules that determines how two computers will communicate with each other. These rules define, for example, how to package data for transfer, how to detect damaged data, and how to optimize throughput. Both computers must use the same protocol for a communications session to succeed.

Simple Protocols

During the early days of telecommunications, people who needed to transfer a file across a phone line or a hardwired asynchronous connection were limited to using text transfer. This is the simplest transfer method, involving only the capturing and transmission of the data stream with no error detection. To receive a file, a buffer is opened to save the information; to send a file, the characters from the chosen file are sent directly out of the communications port to the remote computer.

Of course, no telecommunications connection is perfect, and users soon found that line noise could easily corrupt a file. Thus, file transfer protocols were developed to provide error control. Kermit, Xmodem, Ymodem, Zmodem, and FTP are examples of public domain protocols widely used by computer owners to transfer files. The public domain file transfer protocols are fully described the three chapters following this chapter.

The BLAST Session Protocol

The BLAST Session protocol defines a set of rules for performing file transfer and file management with a remote computer. Under the BLAST Session protocol, three kinds of tasks can be performed:

1. *Files can be transferred between local and remote machines.* The BLAST Session protocol permits files to be transferred bi-directionally—that is, data is sent and received at the same time with automatic error detection and data compression.

2. *Files on the remote machine can be manipulated.* For example, files can be deleted, renamed, or printed on the remote computer. Because these tasks are mediated by the BLAST Session protocol, the commands cannot be garbled by line noise. In addition, the commands are automatically translated into the appropriate instructions on the remote computer. For example, when you give the "List Files" command using the BLAST Session protocol, you will receive a directory listing whether the remote machine is a Macintosh, a VAX, or a computer running the UNIX operating system. You do not need to know the machine-specific instruction.

3. *Messages can be exchanged between the local and remote computer.* Between file transfers, if someone is present at the remote site, you can send messages to and receive messages from the remote operator.

The BLAST Session protocol is much more sophisticated than public domain file transfer protocols. No public domain protocol has all the characteristics of BLAST session protocol. BLAST is generally faster than public domain file transfer protocols because it offers all of the following features:

◊ Bi-directional transfers.

◊ Six levels of compression.

◊ Sliding-window design.

◊ Automatic translation of text files between the local file format and the format of the remote system.

◊ Resumption of interrupted file transfer from the point of interruption.

◊ Security for validating remote users.

BLAST Protocol Design

Bi-Directional and Sliding-Window Capability

The BLAST protocol is capable of transmitting and receiving data packets simultaneously. This simultaneous bi-directional transfer saves time and online charges when files need to be both sent and received.

BLAST operates efficiently over circuits with high propagation delays (the length of time from when a character is transmitted to the time it is received). This resistance to delays is due to BLAST's sliding-window design.

The size of a window is the number of packets that can be sent to the remote computer without BLAST's having to wait for an acknowledgement from the remote. As the remote computer sends acknowledgements, the window slides so that more packets can be sent. For example, if the window size is set to `16`, and the first 4 of 12 packets sent have been acknowledged, the window slides to allow 8 more packets to be sent. In this way, a continuous stream of packets can be sent without BLAST's having to wait for an acknowlegement. The window size and frequency at which acknowledgements are requested can be specified by the user.

These two features—simultaneous bi-directional transfer and sliding-window design—combine to make BLAST a great time saver for long-distance callers. For example, BLAST can upload daily production figures to a host computer over a noisy telephone line at the same time that it downloads the next day's production quotas.

CRC Error Detection

BLAST protocol uses the industry-standard CCITT CRC-16 technique for detecting altered data packets. This is the same method used in IBM SNA/SDLC networks and X.25 packet-switching networks.

Optimized Acknowledgements

When packets of data are transmitted, they must be acknowledged by the receiving computer so that the sender knows that the transfer is complete and accurate. When data is being transmitted in only one direction, the BLAST protocol uses a minimal number of acknowledgement packets flowing in the opposite direction. When data is being transferred in both directions, the data and acknowledgement packets are combined into a single packet. This efficient use of packets is important when working with networks because network charges are often computed on a per-packet rather than a per-byte basis.

Adjustable Packet Size

The BLAST packet size can be set from `1` to `4085` bytes according to the quality and type of connection. A small size minimizes the amount of data that must be retransmitted if line noise is a problem. With high quality connections or with error-detecting modems, packet size can be increased to reduce transmission overhead. Packet size can also be set to optimize network packet utilization.

BLAST Protocol Circuit Requirements

BLAST is flexible in its circuit requirements. Because BLAST does not use any of the ASCII control codes, it is compatible with the use of these control codes for other purposes. For example, BLAST can be employed on circuits where software flow control (CTRL Q/CTRL S) is in use. The XON/XOFF Pacing setup field allows the user to control whether or not BLAST uses this feature. This is very important for load sharing on network virtual circuits and time-shared minicomputers.

BLAST can operate on 7-bit or 8-bit circuits. 7-bit operation allows BLAST to communicate with parity. This does not inhibit BLAST's ability to transmit binary data—you may transfer either 7- or 8-bit data over both 7- and 8-bit circuits.

When using BLAST to communicate with computers that require 7-bit circuits, the setup parameter 7-Bit Channel must be set to YES. This setting slows the throughput of the transfer.

Starting BLAST on a Multi-User System

There are three ways to start a BLAST Session on a remote multi-user computer. Note that you should already be logged into the remote system and appropriate directory.

Manual Method

◊ Select Terminal from the Online menu.

◊ Type the appropriate commands to the remote computer to start a BLAST session. For UNIX, this would be:

  blast -h

  at the command line.

◊ You should see either one of two messages from the remote:

  ;starting BLAST protocol.

  or

  ppp... (only for earlier versions of BLAST)

After the message appears, press *ATTN ATTN* to exit Terminal mode; then select Filetransfer from the Online menu.

Interactive Automatic Method

Select Filetransfer from the Online menu. Your system will automatically start the BLAST session on the remote system.

NOTE: The type of multi-user remote operating system must be identified in the System Type setup field for this method to work. BLAST will then know which automation information to retrieve from the systems.scr library program.

BLASTscript Automatic Method

◊ Write a BLAST script that includes the `FILETRANSFER` statement. This script can be executed from the command line or the Online menu.

◊ `FILETRANSFER` starts a BLAST Session on the remote system and initiates the BLAST Session locally.

NOTE: The type of multi-user remote operating system must be identified in the System Type setup field for this method to work. BLAST will then know which automation information to retrieve from the systems.scr library program.

Starting BLAST on a PC or Other Single-User Computer

If the remote computer is a single-user system, such as a PC, you may start the BLAST Session in one of three ways:

Assisted Method

◊ Select Connect from the Online menu.

◊ Select Filetransfer from the Online menu.

◊ Have the operator on the remote machine select Filetransfer from the BLAST menu.

After the session has started, you can control both BLAST sessions from your keyboard; therefore, the remote operator is no longer necessary. In order for you to be able to complete all transfers and end the session without remote assistance, however, the remote operator must press *CANCEL* before leaving so that the remote system will terminate the session on your command.

Unattended Method

◊ Run the BLAST script slave.scr (found on your distribution media) on the remote system. This script places the remote in "slave" mode, waiting for incoming calls.

◊ Select the Online menu Connect command.

◊ When connected, you have ten seconds to select Filetransfer from the Online menu. If Filetransfer is not selected within this time, the slave assumes the call is not for BLAST, hangs up the modem, and resets for the next call. When the remote receives your Filetransfer command, it automatically initiates the BLAST Session.

BHOST

◊ Run BHOST on the remote system if the remote system is a PC running DOS. BHOST occupies less than 100K of RAM and performs file transfers in background mode.

◊ After establishing a connection with the BHOST machine (see "Connecting to the Host PC" on page 324), select Filetransfer from the Online menu. BHOST will automatically complete the protocol link.

Automatic Filetransfer Handshaking

While entering Filetransfer mode, the two computers will communicate for a few seconds on their own—they will “shake hands” by exchanging information. During handshaking, your system will:

◊ Send its BLAST version and type to be displayed and logged at the other end.

◊ Exchange filetransfer and communication parameters with the remote computer and adjust itself to the other machine’s lowest setup values. For instance, if your setup specifies a Packet Size of 256 bytes and the remote computer is set to 2048, then the lower value of 256 will be used.

◊ Display the Filetransfer menu and an initial assessment of communication line quality.

This process can fail if it does not occur within the time period specified in the Logon Timeout setup field. If handshaking fails, BLAST displays “Logon Timeout” and returns to the Online menu.

BLAST Protocol Timeouts

There are two types of timeouts in BLAST protocol: the Logon Timeout and the Inactivity Timeout. Both timeout values can be specified in fields in the BLAST Protocol Setup (see page 84).

The Logon Timeout is the maximum time in seconds after initiating the BLAST Session protocol that BLAST will wait for the initial handshake with another system. The default value is `120`. If a Logon Timeout exists and the maximum time specified to establish the BLAST Session elapses, BLAST will return to the Online menu.

If the Logon Timeout is set to `0`, the timeout is disabled. Setting the Logon Timeout to `0` at the remote site could “lock up” the remote system; however, BLAST allows you to force a disconnect by following these steps:

◊ Select the Terminal command to enter Terminal mode.

◊ When you see the BLAST message

;starting BLAST protocol.

on the display, type:

```
;DISC.
```

This tells BLAST on the remote system to abort its attempt to enter a BLAST session. Because the message you type will not be echoed on the screen, repeat it several times if necessary. Note that the command is case-sensitive.

The Inactivity Timeout is the maximum time in seconds allowed between the transmission of valid BLAST protocol transfer packets. The default is `120` seconds. If BLAST times out, it will return to the Online menu. A setting of `0` disables the timeout.

NOTE: Using the Local menu during a file transfer suspends transfer activity, causing Filetransfer mode to terminate if the Inactivity Timeout interval is exceeded.

Ending a BLAST Session

The BLAST Session can be terminated in one of four ways:

Normal Menu Escape

Press *CANCEL* at the Filetransfer menu or include an `ESC` statement in a BLAST script to end a filetransfer session.

◊ The files queued for transmission and the files currently being processed complete transmission normally.

◊ The computers complete an exit handshake, and display normal end messages.

◊ Control passes to the Online menu or to the BLASTscript statement following the `ESC`.

NOTE: For completion of the exit handshake, the remote operator must have pressed *CANCEL* unless the remote system is in host mode or is running a script with an `ESC` statement, in which case the remote system will automatically recognize your command.

Single-Attention Abort

Press the *ATTN* key once to quit an interactive transfer or to abort a BLAST script performing a file transfer.

◊ The files queued for transmission will not be sent, and the file currently being transmitted will be marked on the receiving side as interrupted.

◊ The computers complete an exit handshake and display normal end messages.

◊ Control passes to the Online menu or to the BLAST script.

Double-Attention Abort

Press the *ATTN* key twice to quit immediately.

◊ The files queued for transmission will not be sent, and the file currently being transmitted will be marked on the receiving side as interrupted.

◊ The computers do not complete an exit handshake.

◊ The remote is left to time out on its own. You may force a disconnect by typing `;DISC.` as described earlier.

◊ Control passes to the Online menu or to the BLAST script.

Timeout Abort

If a communications failure causes a timeout, the phone is disconnected, or no activity takes place, both computers send an exit handshake when the timeout value is reached.

Performing Filetransfer Commands

Filetransfer Menu

After the handshaking is completed, BLAST will display the Transfer Status Area and the Filetransfer menu (Figure 6-1 below).

FIGURE 6-1

```
BLAST     Filetransfer            default    /usr                   MENU
Send  Get  Message  Remote  Local  File
... send file(s) to the remote system
— local ——————————————————— opt - % xfer - file size — byte cnt - ln qual -
S: <idle>
R: <message>                                                   good (00)
———————————————————————————————————————————————————— ?-help — ESC-exit
```

The basic functions of a filetransfer session are controlled by the following menu commands:

Send – Sends a file or files to the remote system.

Get – Receives a file or files from the remote system.

Message – Sends a text message of up to 67 characters in length to the remote operator. Simply type the message and press ENTER. The message will be queued for transmission to the remote display following completion of other pending filetransfer commands.

Remote – Performs remote system commands. This option is similar to the Local command but offers limited access to the remote computer. See "BLAST Protocol Remote Menu" on page 118 for more detailed information.

Local – Performs local system commands. This is identical to the Local command available from the Offline and Online menus. See "The Local Menu" on page 58 and the note concerning the Local menu and the Inactivity Timeout under the section "BLAST Protocol Timeouts" on page 105.

File – Executes a transfer command file that can control an entire filetransfer session unattended (see "Transfer Command File" on page 115). This command is valid only for transfers using the BLAST protocol.

Transfer Options

Three transfer options can be used in file transfers via the Filetransfer menu command or a BLASTscript `FILETRANSFER` statement:

`t` specifies text translation from the local file format to the destination system's text file format. This switch should *only* be used with ASCII files—do *not* send binary files using the `t` option.

`o` causes the transmitted file to overwrite an existing file with the same name on the receiving system. This will result in the destruction of the original file on the receiving system, so use this option with caution. An error will result if this option is not used and the file already exists on the receiving system.

`a` appends the transmitted file to the end of an existing file with the same name on the receiving system. If the file does not exist on the receiving system, it will be created.

When using the Filetransfer menu command, you are prompted to type one or more of these letters (`t`, `o`, or `a`) to specify your transfer option(s). In a BLAST script, type the letter(s) on a separate line following the name of the file or files to be transferred. For more on using transfer options in a BLAST script, see "Getting and Sending Files" on page 194.

Sending a File

To send a file,

◊ First, select Send from the Filetransfer menu.

 S

◊ At the prompt:

 enter local filename:

 enter a single filename from the current directory or a path specification with a single filename; you may use wildcards (see the section "Wildcards" on the next page) and file transfer switches (see "File Transfer Switches" on page 111). After doing so, press ENTER.

◊ At the prompt:

 enter remote filename:

 Press ENTER only, type a single filename, or type a "%", and any optional switches.

 By default, BLAST will enter the filename (and path, if specified) as you typed it at the local filename prompt. Pressing ENTER only will transfer the file to the remote system, using the local filename (and path if included with the local filename). Typing a different filename (and path, if necessary) will rename the file when it is created on the remote system. See "File Transfer Templates Using the '%' Character" on page 110 for an explanation of "%".

 Some remote computers will interpret optional file transfer switches sent with the remote filename as file-handling and file-attribute controls. After specifying a remote filename, if any, press ENTER.

◊ At the prompt:

specify transfer options: (t=text, o=overwrite, a=append):

Type any combination of the letters `t`, `o`, and `a` or press ENTER only to specify no options. For a fuller description of transfer options, see the preceding section, "Transfer Options."

If you do not specify any options, the file will be transferred to the remote system byte-for-byte as a binary file. If the file exists on the remote system, the transfer will abort.

After specifying options, press ENTER; you will be returned to the Filetransfer menu, and the transfer will begin. The number of bytes sent will appear, as well as a percentage estimate of the amount of data transferred. When the file transfer completes, a message will be sent to your system.

Getting a File

Receiving a file differs only slightly from sending a file. Press G from the Filetransfer menu. You will be prompted for the remote filename first. Any switches added to the end of the remote filename must be valid for that operating system.

Wildcards

By using the wildcard characters "*" and "?", you can transfer multiple source files with similar names. The source files must reside in the same directory and path. The wildcard specifications are as follows:

? Substitutes for a single character.

* Substitutes for multiple characters.

File Transfer Templates Using the "%" Character

When a "%" is entered in the filename field for the target drive, filename(s) from the source drive are transferred to the target drive without the source drive path specification(s).

IMPORTANT: "%" is REQUIRED for the target filename when the source filename contains a "?" or an "*" or when the source filename includes a path and the target filename does not (that is, the target directory is the current working directory).

Some examples are:

Source Name	Target Name	Result
test1.asc	C:\test1.asc	one file in the current source directory, sent to the target (DOS) directory C:\
/tst/test1.asc	%	one file in the source directory /tst, sent to the current target directory
/tst/test1.asc	/tst/test1.asc	one file in the source directory/tst, sent to the target directory /tst (/tst must exist in the current target directory)
test?.asc	%	multiple files in the current source directory—for example, test1.asc, test2.asc, and test3.asc—sent to the current target directory, retaining their source names
test1.*	%	multiple files in the current source directory—for example, test1.asc, test1.lst, and test1.txt—sent to the current target directory, retaining their source names
*	/bin/%	all files in the current source directory sent to the target directory/bin/, retaining their source names.

File Transfer Switches

Instead of specifying transfer options at the prompt, you can append the appropriate file transfer switches to both the local and remote filename specifications. Some remote computers will recognize switches sent with the remote filename as file-handling and file-attribute controls. Experiment with the transfer switches until you obtain the correct results. The valid switches are:

/APP	Append to a file with the same name, if it exists.
/COMP=*n*	Switch compression level value from the value in the compression field of the setup. Use the /COMP=*n* switch at the end of the filename where

n equals the level of compression (0–6). Setting the level to 0 turns off compression.

/FOLLOW=*nn* — Allow data to be transferred from files to which data is being continuously or periodically appended. The /FOLLOW switch is appended to the local filename if being sent, or to the remote filename if being received.

10.7x	For /FOLLOW=*nn*, *nn* specifies the amount of time in seconds that BLAST will wait before checking for an end-of-file marker when transferring a file that is being continuously updated. When /FOLLOW is used and BLAST detects an end-of-file marker for the file being transferred, the file's creation date and time are examined to see if they are set to the operating system's creation date and time. If so, BLAST will wait for the /FOLLOW timeout value before attempting to read the file again. If the date and time are set to any other valid date and time, normal end-of-file processing will occur.
10.8x	BLAST will transfer additions to the file as they occur until *nn* seconds have elapsed from the last addition to the file.

/FWD — Delete file from sending system if the transfer was successful. The /FWD switch is disabled by default. To enable it, toggle the Enable /FWD and /STR setup field (page 88) in the BLAST Protocol subwindow to YES. For the /FWD switch to work, it must be enabled on the *sending* system.

NOTE: The /FWD switch is a very powerful feature of BLAST. Because it allows files to be automatically *deleted* from the sending system, always exercise caution when using it.

/GROUP=*nn* — Preserve or set the group of the file where *nn* is an positive decimal integer that specifies the file group ID.

/OVW — Overwrite a file with the same name if it exists. The ability to use the /OVW switch is enabled by default. To disable use of it, toggle the Enable /OVW and Remote Cmds setup field (page 88) in the BLAST Protocol subwindow to `NO`.

NOTE: If use of the /OVW switch is disabled on the receiving system, BLAST protocol will not allow the file to be overwritten.

/OWNER=*nn*	Preserve or set the owner of the file, where *nn* is a positive decimal integer that specifies the file owner ID.
/PERMS=*nnnn*	Preserve or set file permissions where *nnnn* is an octal number that contains the file permissions for the target file. This switch is automatically appended to files sent from the local system and can also be specified by the remote system. See "Permissions" on page 150 and your system documentation for more information about permissions.

`4000`	Set user ID on execution
`20#0`	Set user ID on execution if "#" is `7`, `5`, `3`, or `1` (grant execute permission); enable mandatory locking if "#" is `6`, `4`, `2`, or `0`.
`1000`	Set the sticky bit
`0400`	Read by owner
`0200`	Write by owner
`0100`	Execute (search in directory) by owner
`0040`	Read by group
`0020`	Write by group
`0010`	Execute (search in directory) by group
`0004`	Read by others
`0002`	Write by others
`0001`	Execute (search in directory) by others
`0000`	No permissions

If the account on the receiving system does not have all of the necessary permissions to create the file as specified by this switch, BLAST will create the file with as many permissions as the account allows.

/STR	Delete file from receiving system if transfer was unsuccessful. The /STR switch is disabled by default. To enable it, toggle the Enable /FWD and /STR setup field (page 88) in the BLAST Protocol subwindow to `YES` on the receiving system.

/TXT Perform text translation. BLAST will convert carriage returns, line feeds, and end-of-file markers to the receiving system's text format.

You might, for example, specify text translation and overwriting of an existing file with the following filename:

```
test1.doc/TXT/OVW
```

Or you might specify that the file will be automatically deleted from your system after it has been successfully sent and that it will be sent with a compression level of 6:

```
test1.doc/FWD/COMP=6
```

Filenames Restrictions with BLAST Protocol

With BLAST protocol, you should *not* give a file the same name as a switch since BLAST protocol will assume that the file is a switch and look for a file with the name of the folder containing the file. Thus, the transfer of the file will not occur and you will get an error message. Filenames (uppercase or lowercase) to avoid are: app, comp=*n*, follow=*nn*, fwd, group, ovw, owner=*nn*, perms=*nnnn*, str, and txt (where *n* is a number from 0 to 9).

You can work around this restriction by changing your local and remote working directories to the ones containing the file you want to transfer and giving the filename without a path. To change your local working directory interactively, choose Chdir command from the Local menu. To change your remote working directory interactively, choose the Chdir command from the Remote menu.

Alternatively, you may do a scripting workaround:

```
FILETRANSFER
LCHDIR "/u/Pat/work"    # Change local directory
REMOTE
  Chdir                 # Change working directory
  /usr/customer         # Name of new directory
  ESC
SEND
App                  # Filename only--no path;retain file-
                     # name on remote;no transfer options


ESC
```

If, on the receiving system, you give the file a new name that is not that of a switch, you *can* give a path. For instance, if in the script above, App was given the new name Sales.txt on the receiving machine, you could change the script to the following:

```
FILETRANSFER
LCHDIR "/u/Pat/work"    # Change local directory
Send
App                     # Filename only--no path
/usr/customer/Sales.txt # Give new name and full path

Esc
```

Restarting an Interrupted File Transfer

Disconnections and interruptions in sending long files can be costly and time-consuming. BLAST can restart transfer of files from the point of interruption without having to restart transmission from the beginning of the file.

If a filetransfer session is interrupted and you wish to restart from the point of interruption, both local and remote systems must time out or be interrupted by *ATTN ATTN*. After the session has been interrupted or aborted, you may restart the session by following these steps:

◊ Reconnect, if necessary, and restart the filetransfer session.

◊ Send the EXACT file that was being sent when interrupted.

◊ Do NOT indicate the overwrite or append options.

BLAST restarts from the last point at which its buffers were flushed to disk. This may be right at the interrupt point or as much as 10K before the interrupt point.

NOTE: Adding the /STR switch to a filename eliminates the possibility of resuming an interrupted transfer of that file.

Transfer Command File

A transfer command file is a text file that contains line-by-line instructions describing functions to be performed during a BLAST protocol filetransfer session. Any word processor or editor can create a transfer command file, but it must be saved in text only or

ASCII format under any name that you choose. Transfer command files are also called error-free command files.

A transfer command file can be invoked interactively by selecting the File command from the Filetransfer menu, or from within a BLAST script by using the following BLASTscript commands:

```
FILETRANSFER
FILE
Filename   # name of the transfer command file
ESC
```

If the transfer command file is in the current directory, you only have to specify the filename; if it is in any other directory, you must specify the full path.

The command file contains an unlimited number of commands, each as a separate line of text. Files, messages, and remote system commands can be sent and remote files can be received. Filetransfer commands are entered as one line, with the source and destination specifiers separated by a space. If any file transfer switches are required, they are entered following the file specifier(s).

Command Formats

The text in a transfer command file must begin in the first column of every line. Commands in a transfer command file accomplish one of four tasks:

1) Send a File:

   No special character is required; simply type the name of the local file to send and, separated by a space, the name for the file on the remote system. If no remote name is given, BLAST will use the local name. Any file transfer switches must be typed immediately following the filename:

```
local_filename[switches] [remote_filename[switches]]
```

2) Get a File:

   The first character in the line must be a plus sign (+). Immediately following the "+", enter the name of the file to receive from the remote system and, with no intervening space, any file transfer switches. If a different name is desired for the local file, type a single space after the remote filename and then type the local filename with any switches immediately following:

`+`*`remote_filename`*`[`*`switches`*`] [`*`local_filename`*`[`*`switches`*`]]`

Note that it is more efficient to put all Gets (lines beginning with "+") first, so that the remote file requests queue up on the remote. This allows for true bi-directional transfer during command file operations.

3) Send a Display Message:

The first character in the line must be a semicolon (;). Immediately following the semicolon, type the desired message, which will be transmitted to the remote display and the remote log.

```
;Now Sending Sales Reports
```

4) Send a Command to the Remote System:

The character in the first column must be an exclamation point (!). Immediately following the exclamation point, type one of the following commands:

```
!dir
```

The valid remote operating system commands are:

`DIR`
Display the contents of the current remote directory.

`TYPE` *`filename`*
Type the contents of the specified remote file to the screen.

`C`
Display the next page of a multi-page display.

`PRINT` *`filename`*
Print the specified file on the remote printer.

`REN` *`oldname newname`*
Rename the specified remote file to the new name.

`ERA` *`filename`*
Erase the specified remote file.

`CHDIR` *`path`*
Change from the current remote directory to the specified remote directory.

Example

To understand the use of transfer command files, imagine that a salesman named Joe is using BLAST to keep track of current pricing changes and to send in current orders. He will always get the file called curprice.fil and send the file called joeorder.fil. Joe can create an error-free command file named joe.cmd, which looks like this:

```
;I want to get current price lists
+curprice.fil/txt joeprice.fil/txt/ovw
;Now I am about to send in today's orders
joeorder.fil/txt todayord.fil/txt/ovw
!dir
```

To use this command file, Joe would choose File from the Filetransfer menu and type in the name joe.cmd at the prompt. The following sequence of events then takes place:

◊ The first message in the command file appears on the screen.

◊ The file curprice.fil is retrieved and overwrites the old joeprice.fil.

◊ The second message appears.

◊ Joeorder.fil is sent and overwrites the old todayord.fil.

◊ Finally, the contents of the current directory of the remote computer are displayed on Joe’s screen.

BLAST Protocol Remote Menu

The Filetransfer menu contains a Remote command that takes you to the Remote menu. The Remote menu allows a user with no knowledge of the remote operating system to manage files on that system. For example, a UNIX user can delete a file on a VMS remote system without actually typing theVMS delete command. BLAST will “translate” the command automatically. Remote commands affect only files in the current remote directory unless you specify a pathname.

NOTE: The Enable /OVW and Remote Cmds setup field (page 88) in the BLAST protocol subwindow must be enabled on the remote system in order for you to delete, rename, or print files on the remote system.

Following is a description Remote menu commands:

List – Operates like the Local List command, except that it displays the contents of the current remote directory. You will be prompted to choose either a detailed (long) or non-detailed (short) list and then to specify a filename; you may use a specific filename, a filename with wildcard characters (for example, “*”), or press ENTER to display all files in the current remote directory.

Delete – Deletes a single file or multiple files from the remote system. You may use a specific filename or a filename with wildcard characters (for example, “*”).

Rename – Renames a remote file.

Type – Displays a remote file on the BLAST screen.

Print – Prints a remote file to the remote printer.

Chdir – Changes the current remote directory to one that you name. BLAST will check this directory for any files that you specify with the Remote menu commands.

More – Scrolls a page of data when either the List or Type commands cause more than one full screen of data to be received. You will be prompted to execute the More command in order to see the remaining pages, one page at a time.

Automating the BLAST Session Protocol

The BLAST Session protocol can be fully automated through scripting. For information on writing scripts using the BLAST protocol, see “File Transfers with BLAST Session Protocol” on page 194.

Fine-Tuning the BLAST Session Protocol

Packet Size

Most computers can process packets of 256 characters. Set the Packet Size setup field (page 90) to `256` or higher unless the phone line quality is poor. Small packet sizes reduce the number of bytes requiring retransmission over noisy lines. Computers connected directly by cables will benefit from a much larger packet size, such as `4085`. In a BLAST script, the reserved variable for packet size, `@PAKTSZ`, can be set anytime before entering a filetransfer session.

Compression Levels

BLAST performs automatic data compression during file transfers with the BLAST protocol, reducing the number of characters sent and the transfer time.

Compression level is specified in BLAST Protocol subwindow setup fields (pages 88 – 88). Possible values for Receive Compression Level and Send Compression Level are `0` (no compression) to `6`. The default is `4`, which provides the best performance for average-sized files. Compression can also be selected by the `@RCOMP_LEV` (receive) and `@SCOMP_LEV` (send) BLASTscript reserved variables.

Data compression requires additional RAM during file transfers. The amount of RAM necessary varies with the compression level.

Compression Level 0 – Level `0` specifies that no compression will be used. Choose level `0` when your CPU is slow and the baud rate is high. In this situation, the overhead needed for compression can actually increase transfer time.

IMPORTANT: Always use compression level `0` when transferring pre-compressed files.

Compression Level 1 – Use level `1` when your data has strings of duplicate characters. Such data could include row and column reports, which have many embedded blanks, and executable files with blocks of nulls. In some cases, compression level `1` improves performance over high-speed modems with hardware data compression enabled.

Compression Level 2 – Starting with level `2`, compression requires more work by both computers. With a standard modem and two fast machines, however, levels `2`–`4` will save transmission time.

Compression Level 3 and 4 – Levels `3` and `4` of compression are most effective when a limited character set is used or there are repetitious patterns. Because spreadsheets and databases have many repetitious patterns and a limited character set, they are highly compressible.

Compression Level 5 and 6 – Levels 5 and 6 compression are most effective for very large files (above 500 K). On large files (above 500K), the receiving computer may notice a significant delay before the first block is received while the sending computer calculates maximum compression.

Filetransfer Security with BLAST Protocol

Disabling File Overwrites and Remote Commands

The Enable /OVW and Remote Cmds setup field (page 88) and the script variable `@ENABLERCMD` (page 272) control whether or not remote commands and file overwrites are allowed during Filetransfer mode. Note that disabling /OVW affects only *local* files. For example, you will still be able to send a file with the /OVW switch because the file will be overwritten on the *remote* system.

Disabling the /FWD and /STR Switches

The Enable /FWD and /STR setup field (page 88) and the `@ENABLEFS` (page 271) script variable control whether or not the /FWD and /STR file transfer switches are allowed during Filetransfer mode. Note that disabling these switches affects only *local* files. For example, you will still be able to get a file with the /FWD switch because the successfully transferred file will be deleted from the *remote* system. See “File Transfer Switches” on page 111.

NOTE: Adding the /STR switch to a filename eliminates the possibility of resuming an interrupted transfer of that file.

Using the Transfer Password

If you have limited a remote user’s access so that BLAST automatically run’s a specific BLAST setup when a user logs into your system (see “Limiting Access” on page 154), you can insure additional security by specifying a Transfer Password for that setup. Without the password, the remote user may only send and receive messages while in Filetransfer mode. The Transfer Password can be set by entering it into the Transfer Password setup field (page 87) or by setting the reserved variable `@TRPASSWD` (page 290) in a slave script on the remote system.

NOTE: The transfer password is superseded by the Secure BLAST password (see “Using Secure BLAST” on page 155).

After entering a filetransfer session, the remote user must send the transfer password to the host machine using the Send command from the Filetransfer menu or a `FILETRANSFER` statement in a script. If the user issues a Send command from the Filetransfer menu, the following special format for the local filename must be used:

!password=*your_password*

where *your_password* represents the password stored on the host system. The remote filename field is left blank as are the text, overwrite, and append options. If the correct password is successfully sent, the remote user will see a message stating that the password has been validated. The password must be typed exactly as it is set on the host system!

If a BLAST script is used, the same special local filename format must be sent to the host computer, for example:

```
FILETRANSFER
SEND
!password=blue2


SEND
myfile.rpt
yourfile.rpt
ta
ESC
```

Because the remote filename and send transfer options are not used, two blank lines must follow the `!password=`*your_password* statement. See "Getting and Sending Files" on page 194 for information on scripting file transfers.

Since the remote user has to enter the password through BLAST interactively or through a script, the use of Transfer Password deters an unauthorized user from breaking your security by submitting a rapid series of passwords.

NOTE: The Transfer Password is intended to validate remote users logging onto your system. If a local operator uses a setup with a Transfer Password entered, he or she will not be able to receive files without the remote user sending the password.

Chapter 7

FTP with 10.8x

Introduction

BLAST includes FTP for transferring files with systems via TCP networks.

To choose FTP, select FTP from the Protocol setup field (page 84) or set the reserved variable `@PROTOCOL` (page 281) to FTP in a script.

Starting an FTP Session

If you have selected FTP as your protocol, choose Filetransfer from the Online menu—BLAST will automatically log into the remote system using the values from the Userid and Password setup fields.

FTP Filetransfer Menu

You will notice from the screen shown in Figure 7-1 below that the FTP Filetransfer menu is slightly different from the menu displayed during a BLAST protocol session. Following is a brief description of the command options of this menu:

Send – Sends a file to the remote system.

Get – Retrieves a file from the remote system.

Local – Performs local system commands. This command takes you to the Local menu. Note that all filetransfer activity is suspended while you are using the local system.

Remote – Performs remote system commands. This option allows a user with no specific knowledge of the remote operating system to manage remote files. For example, a user can delete a file without actually typing the delete command of the remote operating system (see “FTP Remote Menu” on page 128).

FIGURE 7-1

```
BLAST     FTP                         default    /usr                        MENU
Send  Get  Local  Remote
... send file(s) to the remote system
— local ————————————————————————— % xfer - file size — byte cnt - retries -
S: <idle>
R: <idle>
——————————————————————————————————————————————————————————— ?-help — ESC-exit







-----------------------<< Entering FTP Transfer Mode >>----------------------
```

Sending and Receiving Files with FTP

The following two sections describe interactive file transfers. For a discussion of scripting FTP file transfers, see “File Transfers with FTP Using 10.8x” on page 197.

Sending Files with FTP

To send a file,

◊ Select Send from the Filetransfer menu.

◊ At the prompt:

enter local filename:

enter a filename from the current directory or a filename with a fullpath. You may use wildcards (see “Wildcards” on page 110) and any supported switches (see “File Transfer Switches with FTP” on page 126). After doing this, press ENTER.

◊ At the prompt:

enter remote filename:

Press ENTER only, type a single filename, or type a “%”, and any optional switches.

By default, FTP will enter the filename (and path, if specified) as you typed it at the Local Filename prompt. Pressing ENTER only will transfer the file to the remote system using the local filename (and path if included with the local filename). Typing a different filename (and path, if necessary) will rename the file when it is created on the remote system. Alternatively, you may use a file transfer template using the “%” (see “File Transfer Templates Using the “%” Character” on page 110). For a list of supported switches, see “File Transfer Switches with FTP” on page 126.

When the FTP transfer completes, a message will be sent to your system and you will be returned to the Filetransfer menu.

Getting Files with FTP

Receiving files with FTP differs only slightly from sending files.

◊ Press G from the Filetransfer menu.

◊ At the prompt, enter the remote filename or filename with full path; you may use wildcards (see “Wildcards” on page 110).

◊ At the prompt, enter the local filename, filename with full path, or file transfer template (see “File Transfer Templates Using the

"%" Character" on page 110), and any supported switches (see "File Transfer Switches with FTP" on page 126). When the FTP transfer completes, a message will be sent to your system and you will be returned to the Filetransfer menu.

NOTE: FTP `GET`s should be used with caution. In the FTP protocol, the markers for end-of-file and for close-connection are the same. Thus, incomplete file receives resulting from connection failures are reported as successful file transfers in both the File Transfer Status Area and the log file.

File Transfer Switches with FTP

FTP supports the file transfer switches listed below; all other file transfer switches are ignored by FTP:

/APP	Append to a file with the same name if it exists. Without this switch, a file is automatically overwritten.
/FWD	Delete file from sending system if transfer was successful. **NOTE:** The /FWD switch is a very powerful feature of FTP. Because it allows files to be automatically *deleted* from the sending system, always exercise caution when using it.
/STR	Delete file from receiving system if transfer was unsuccessful.
/TXT	Perform text translation. BLAST will convert carriage returns, line feeds, and end-of-file markers to the receiving system's text format.

You might, for example, specify appending and text translation of an existing file with the following filename:

```
test1.doc/APP/TXT
```

Filenames Restrictions with FTP

With FTP, you should *not* give a file the same name as a switch since FTP will assume that the file is a switch and either ignore it (if the switch is unsupported by FTP) or look for a file with the name of the folder containing the file (if the switch is supported by FTP). In either case, the transfer of the file will not occur and you will get an error message. Filenames (uppercase or lowercase) to avoid are: app, comp=*n*, follow=*nn*, fwd, group, ovw, owner=*nn*, perms=*nnnn*, str, and txt (where *n* is a number from `0` to `9`).

You can work around this restriction by changing your local and remote working directories to the ones containing the file you want to transfer and giving the filename without a path. To change your local working directory interactively, choose Chdir command from the Local menu. To change your remote working directory interactively, choose the Cwd command from the Remote menu.

Alternatively, you may do a scripting workaround:

```
FILETRANSFER
LCHDIR "/u/Pat/work" # Change local directory
REMOTE
  Cwd                # Change working dir
  /usr/customer      # Name of new directory
  ESC
SEND
App                  # Filename only--no path

ESC
```

If, on the receiving system, you give the file a new name that is not that of a switch, you *can* give a path. For instance, if in the script above, App was given the new name Sales.txt on the receiving machine, you could change the script to the following:

```
FILETRANSFER
LCHDIR "/u/Pat/work"    # Change local directory
Send
App                     # Filename only--no path
/usr/customer/Sales.txt # Give new name and full path
Esc
```

Ending an FTP Session

FTP sessions end automatically when all specified files are transferred and you press ESC.

FTP Remote Menu

The Remote menu allows a user with no knowledge of the remote operating system to do limited file management on the remote system. Following is a brief description of the three command options of the FTP Remote menu:

List – Operates like the Local List command, except that it displays the contents of the remote current directory. You will be prompted to choose either a detailed (long) or non-detailed (short) list and then to specify a filename, a filename using wildcard characters (see “Wildcards” on page 110), or all files.

Delete – Deletes a single file or multiple files from the remote system. You may use a specific filename or a filename with wildcard characters (for example, “*”).

Cwd – Changes the server’s working directory. You will be prompted for the new directory name.

Chapter 8

Kermit Protocol

Many communication products support Kermit protocol on a wide range of computers, but there are different versions of Kermit, two of which BLAST supports. The simplest version is a file transfer program that requires commands to be entered at both the sending and receiving computers (using the Send and Receive commands). The more sophisticated version is the Kermit server. The Kermit server accepts commands from a remote user and performs specified operations (using the Send, Get, and Remote commands).

Kermit Filetransfer Menu

You will notice from the screen shown in Figure 8-1 on the next page that the Kermit Filetransfer menu is slightly different from the menu displayed during a BLAST protocol session. Below is a brief description of the command options of this menu.

Send – Sends a file to a Kermit program. You will be prompted for the local and remote filenames.

Get – Receives a file from a Kermit server. You will be prompted for the remote and local filenames.

FIGURE 8-1

```
BLAST     Kermit                   default    /usr                        MENU
Send  Get  Receive  reMote  Finish  Bye
... send file(s) to the remote system
— local ————————————————————————— % xfer - file size — byte cnt - retries -
S: <idle>
R: <idle>
———————————————————————————————————————————————————————— ?-help — ESC-exit



-----------------------<< Entering KERMIT Transfer Mode >>---------------------
```

Receive – Receives a file from a simple Kermit. You must specify a local filename.

Remote – Performs remote Kermit server commands. This option allows a user with no specific knowledge of the remote operating system to manage its files. For example, a user can delete a file without actually typing the delete command of the remote operating system (see “Kermit Remote Menu” on page 134).

Finish – Returns you to the Online menu. Kermit server finishes transfer and exits without logging off; thus, you may continue the session.

Bye – Ends Kermit server mode *and* logs off of the remote system. Depending on the remote modem settings, the connection may or may not be broken. You will be returned to the Online menu.

NOTE: Once you begin Kermit server, you can continue to do file transfers until you exit the server by selecting Finish or Bye from the Filetransfer menu.

Sending and Receiving Files with Kermit

The following two sections describe interactive file transfers. For a discussion of scripting Kermit file transfers, see “File Transfers with Kermit” on page 197.

Sending Files with Kermit

Following are directions for sending a file to a remote computer:

Kermit Server

◊ In Terminal mode, begin the Kermit program on the remote system.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Send command. You will be prompted for the local and remote filenames. For the local filename, you may enter a single filename from the current directory or a path specification with a single filename. You may use wildcards (see "Wildcards" on page 110), but you *cannot* use file transfer switches.

◊ The transfer will begin, and the number of bytes sent will be displayed in the File Transfer Status Area.

Simple Kermit

◊ In Terminal mode, begin the simple Kermit program on the remote system.

◊ In simple Kermit on the remote system, issue a receive command.

◊ Exit Terminal mode, select Filetransfer, and then select Send. You will be prompted for local and remote filenames. If you designate a remote filename with the simple Kermit receive command, a filename entered at the remote filename prompt will be ignored.

Receiving Files with Kermit

BLAST's implementation of Kermit supports both the Kermit server Get command and the simple Kermit Receive command to transfer files from a remote computer. Following are directions for transfers from a remote computer:

Kermit Server

◊ In Terminal mode, begin the Kermit server program on the remote system.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Get command. You will first be prompted for the remote filename—you may enter a single filename from the current directory or a path specification with a single filename; you may include wildcards (see “Wildcards” on page 110). You will then be prompted for a local filename. Optionally, you may add any supported file transfer switches (see “File Transfer Switches with Kermit” on page 132). Once you have entered the filenames and any switches, the transfer request is automatically sent to the remote.

◊ Unless you specify otherwise, the received file will be saved to your current directory.

NOTE: If you have an existing file with the same name, the file will be renamed when the Warning setup field (page 92) is set to `ON`. When this field is set to `OFF`, the existing file will be automatically overwritten.

Simple Kermit

◊ In Terminal mode, begin the simple Kermit program on the remote system.

◊ In Kermit on the remote system, send the file by invoking the send command.

◊ Exit Terminal mode, select Filetransfer, and then select Receive. You will then be prompted for a local filename; optionally, you may add any supported file transfer switches (see the next section “File Transfer Switches with Kermit”).

◊ Unless you specify otherwise, the received file will be saved to your current directory.

NOTE: If you have an existing file with the same name, the file will be renamed when the Warning setup field (page 92) is set to `ON`. When this field is set to `OFF`, the existing file will be automatically overwritten.

File Transfer Switches with Kermit

Kermit ignores all file transfer switches on sending filenames and supports the following file transfer switches on receiving filenames:

/APP	Append to a file with the same name if it exists.
/GROUP=*nn*	Preserve or set the group of the file where *nn* is a positive decimal integer that specifies the file

	group ID. Note that on some systems only the user root can change the group attribute.
/OVW	Overwrite a file with the same name if it exists.
/OWNER=*nn*	Preserve or set the owner of the file, where *nn* is a positive decimal integer that specifies the file owner ID. Note that on some systems only the user root can change the owner attribute.
/PERMS=*nnnn*	Preserve or set file permissions where *nnnn* is an octal number that contains the original file permissions. This switch is automatically appended to files sent from the local system and can also be specified by the remote system. See “Permissions” on page 150 and your system documentation for more information about permissions.

`4000`	Set user ID on execution
`20#0`	Set user ID on execution if “#” is `7`, `5`, `3`, or `1` (grant execute permission); enable mandatory locking if “#” is `6`, `4`, `2`, or `0`.
`1000`	Set the sticky bit
`0400`	Read by owner
`0200`	Write by owner
`0100`	Execute (search in directory) by owner
`0040`	Read by group
`0020`	Write by group
`0010`	Execute (search in directory) by group
`0004`	Read by others
`0002`	Write by others
`0001`	Execute (search in directory) by others
`0000`	No permissions

If the account on the receiving system does not have all of the necessary permissions to create the file as specified by this switch, BLAST will create the file with as many permissions as the account allows.

Filenames Restrictions

With Kermit Protocol, you should *not* give a file the same name as a switch since BLAST will assume that the file is a switch and either ignore it (if the switch is unsupported by Kermit) or look for a file with the name of the folder containing the file (if the switch is supported by Kermit). In either case, the transfer of the file will not occur and you will get an error message. Filenames (uppercase or lowercase) to avoid are: app, comp=*n*, follow=*nn*, fwd, group, ovw, owner=*nn*, perms=*nnnn*, str, and txt (where *n* is a number from `0` to `9`).

You may work around this restriction by changing your local current and remote current directory to the one containing the file you want to transfer and giving the filename without a path. To change your local working directory interactively, choose the Chdir command of the Local menu. To change your remote directory interactively using Kermit server, choose Remote from the Kermit Filetransfer menu and then select the Cwd (Change Working Directory) command. To change your remote directory interactively using simple Kermit, access Terminal mode and give the "change current directory" command for that operating system.

Alternatively, you may do a scripting workaround. To change the local working directory, use the `LCHDIR` command. To change the remote working directory using the Kermit server, issue a `FILETRANSFER/REMOTE/Cwd` multi-line command statement. To change the remote working directory using simple Kermit or Kermit server, `TSEND` a "change working directory" command to the remote. For example, the following script fragment changes the current remote directory on a UNIX machine to /u/sales.

```
TSEND "cd /u/sales", CR
```

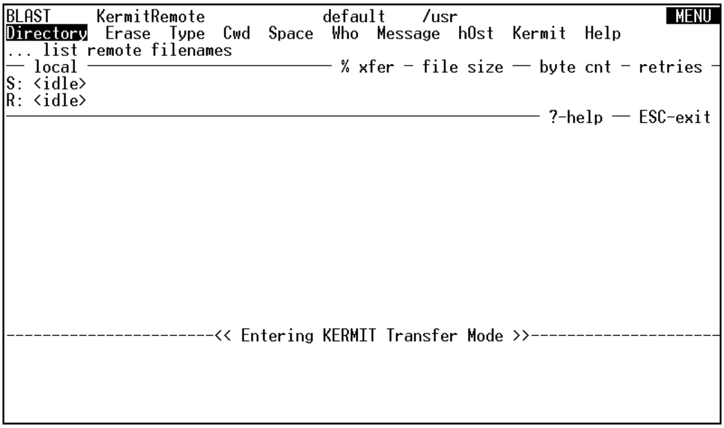
See "File Transfers with Kermit" on page 197 for more on scripting for Kermit.

Kermit Remote Menu

Notice that the Kermit Remote menu (Figure 8-2, next page) offers a selection of commands different than those of the BLAST protocol. These functions operate on the remote system in Kermit server mode. Unreliable results can occur, however, if you use a command that is not directly supported by the server. The Remote menu commands are:

Directory – Displays the **s**erver's current working directory or a directory you specify; wildcards can be used.

FIGURE 8-2



Erase – Deletes a file in the server's current working directory or in a directory you specify by giving the full path of the file; wildcards can be used.

Type – Displays a remote file on your screen. Kermit does not support a page pause, so you must use CTRL S to pause and CTRL Q to resume the flow of text.

Cwd – Changes the server's working directory. You will be prompted for the new directory name.

Space – Displays the server's free drive space.

Who – Displays users currently logged onto the remote. If you specify a user name, information on that name only will appear.

Message – Sends a one-line message to be displayed to the remote operator.

Host – Sends an operating system command to the remote. The command is executed immediately.

Kermit – Sends a Kermit language command to modify session parameters, for example, `SET FILE TYPE BINARY`.

Help – Displays a short list of the commands currently available on the Kermit server. Because servers can support different commands, the Help command can be a valuable reminder of what is available through the Kermit server.

The Kermit `DISABLE` command can lock most of these menu commands. For example, the command `DISABLE ERASE` will prevent files from being deleted on the remote system.

Chapter 9

Xmodem, Ymodem, and Zmodem Protocols

BLAST includes the public domain protocols Xmodem, Ymodem, and Zmodem for transferring files as an alternative to BLAST protocol.

Before choosing Xmodem, Ymodem, or Zmodem for a major application, ask yourself:

◊ Will you need to transfer files with computers using other operating systems?

◊ Do your transfers need to be fast and 100% error free?

◊ Do you want the ability to execute commands on the remote system without special knowledge of the command syntax?

If you have answered “Yes” to any of these questions, you should use BLAST protocol on your remote system if it is available; Xmodem, Ymodem, and Zmodem protocols do not support both near-transparent remote access to other operating systems nor fast, 100% error-free transfers.

The following instructions are very general. Actual procedures for using Xmodem, Ymodem, and Zmodem will vary depending on the implementation of these protocols on the remote system. Many communications products support the standard implementation of these protocols; nevertheless, you should be aware that there are different, incompatible versions that might not work successfully with BLAST.

Command Line Features

If you have chosen the Xmodem or Ymodem protocol in your setup, you can specify an end-of-transmission (EOT) timeout parameter using a command line switch in the following format:

10.7x	blast -y*number*
10.8x	blast -e*number*

where timeout is equal to *number*/100 seconds. The minimum timeout is .1 second (10) and the maximum is 60 seconds (6000). For example, -y1111 or -e1111 sets the timeout to 11.11 seconds.

10.8x	You can also select the pad character for Xmodem using the following format: blast -p*x* where *x* specifies the character expressed as a hexadecimal value. For example, *-p21* specifies “21” as the pad character. The -h command line switch may also be used for Xmodem, Ymodem, and Zmodem file transfers from a remote system not running BLAST. See “BLAST Operation as a Pseudohost With 10.8x” on page 204 for details.

Invoking a command line parameter affects these protocols only for the duration of that communications session.

BLAST supports Xmodem1K CRC as well as Xmodem CRC and the standard Xmodem checksum protocol. When you select Xmodem as your protocol, BLAST will automatically determine which implementation of Xmodem is on the remote system and choose the correct counterpart on your local system.

10.8x	You may change your error-detection setting through the Error Detection setup field (page 94) of the Xmodem protocol subwindow.

NOTE: Xmodem is only compatible with 8-bit connections.

The following two sections describe interactive file transfers. For a discussion of scripting Xmodem file transfers, see “File Transfers with Xmodem and Xmodem1K” on page 200.

Sending Files with Xmodem

To send a file using Xmodem:

◊ In Terminal mode, begin the Xmodem or Xmodem1K receive program on the remote computer, specifying a filename if needed.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Send command. You will be prompted for the local filename.

10.8x	Optionally, you may add the /TXT file transfer switch (see “File Transfer Switches Using 10.8x with Xmodem” on page 140).

Receiving Files with Xmodem

To receive a file using Xmodem:

◊ In Terminal mode, begin the Xmodem or Xmodem1K send program on the remote computer.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Get command. You will be prompted for the filename. If the file already exists on the local machine, you will get an error message.

10.8x	Optionally, you may add any supported file transfer switches (see “File Transfer Switches Using 10.8x with Xmodem” on page 140). For example, you may overwrite an existing file (and avoid an error message) by adding the /OVW switch to the local filename when prompted for the name.

File Transfer Switches Using 10.8x with Xmodem

With Professional UNIX 10.8x, Xmodem supports several file transfer switches; it ignores all switches that it does not support.

File Transfer Switches for 10.8x Using Xmodem		
/APP	Receive	Append to a file with same name if it exists.
/FWD	Send	Delete file from sending system if the transfer was successful.
/GROUP=*nn*	Receive	Preserve or set the group of the file where *nn* is a positive decimal integer that specifies the file group ID. Note that on some systems only the user root can change the group attribute.
/OVW	Receive	Overwrite a file with same name if it exists.
/OWNER=*nn*	Receive	Reserve or set the owner of the file, where *nn* is a positive decimal integer that specifies the file owner ID. Note that on some systems only the user root can change the owner attribute.
/PERMS=*nnnn*	Receive	Preserve or set file permissions where *nnnn* is an octal number that contains the original file permissions. This switch is automatically appended to files sent from the local system and can also be specified by the remote system. See “Permissions” on page 150 and your system documentation for more information about permissions `4000` Set user ID on execution `20#0` Set user ID on execution if “#” is `7`, `5`, `3`, or `1` (grant execute permission); enable mandatory locking if “#” is `6`, `4`, `2`, or `0`. `1000` Set the sticky bit `0400` Read by owner

		`0200` Write by owner `0100` Execute (search in directory) by owner `0040` Read by group `0020` Write by group `0010` Execute (search in directory) by group `0004` Read by others `0002` Write by others `0001` Execute (search in directory) by others `0000` No permissions If the account on the receiving system does not have all of the necessary permissions to create the file as specified by this switch, BLAST will create the file with as many permissions as the account allows.
/STR	Receive	Delete file from receiving system if transfer was unsuccessful.
/TXT	Send	Send file as ASCII using the value stored in `@XYRLTS`.
	Receive	Receive file as ASCII using the value stored in `@XYRLTR`.

Ymodem Protocol

BLAST supports the standard Ymodem and Ymodem G protocols. *Do not use Ymodem G protocol unless there are properly configured error-correcting modems on both ends of the connection*.

.

The following two sections describe interactive file transfers. For a discussion of scripting Ymodem file transfers, see “File Transfers with Ymodem and Ymodem G” on page 201.

Sending Files with Ymodem

To send a file using Ymodem:

◊ In Terminal mode, begin the Ymodem or Ymodem G receive program on the remote computer.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Send command. You will be prompted for the filename. You may enter a single filename from the current directory or a path specification with a single filename; you may use wildcards (see "Wildcards" on page 110).

10.8x	Optionally, you may add the /TXT file transfer switch. You will not be able to add any other file transfer switches (see "/TXT Switch Using 10.8x with Ymodem" below).

Receiving Files with Ymodem

To receive a file using Ymodem:

◊ In Terminal mode, begin the Ymodem or Ymodem G send program on the remote computer.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Get command. The transfer will begin immediately *without* prompting for a local filename.

File Transfer Switches Using 10.8x with Ymodem

With Professional UNIX 10.8x, Ymodem cannot set switches on receiving filenames and ignores all switches on sending filenames except the /TXT, which specifies that the file be sent as ASCII using the value stored in `@XYRLTS` (page 300), and the /FWD switch.

Zmodem Protocol

BLAST supports the standard Zmodem protocol in both single-file and batch modes. BLAST also supports a variety of special Zmodem features that can be activated through the setup fields of the Zmodem protocol subwindow (page 94).

The following two sections describe interactive file transfers. For a discussion of scripting Zmodem file transfers, see "File Transfers with Zmodem" on page 203.

Sending Files with Zmodem

To send a file using Zmodem:

◊ In Terminal mode, begin the Zmodem receive program on the remote computer

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Send command. You will be prompted for the filename. You may enter a single filename from the current directory or a path specification with a single filename; you may use wildcards (see “Wildcards” on page 110).

10.8x	Optionally, you may add supported file transfer switches (see “File Transfer Switches Using 10.8x with Zmodem” below).

Receiving Files with Zmodem

To receive a file using Zmodem:

◊ In Terminal mode, begin the Zmodem send program on the remote computer.

◊ Exit Terminal mode, select the Filetransfer command from the Online menu, and then select the Get command. The transfer will begin immediately *without* prompting for a local filename.

NOTE: If the Auto Receive setup field (`@ZMAUTODOWN`) is set to `YES`, you do not have to select the Get command; Zmodem transfers the file automatically when you enter Filetransfer mode.

File Transfer Switches Using 10.8x with Zmodem

With Professional UNIX 10.8x, Zmodem supports several file transfer switches for sending filenames (see table below). Zmodem cannot set switches on receiving filenames and ignores all unsupported switches.

File Transfer Switches for 10.8x Using Zmodem		
/APP	Send	Specify APPEND as File Management option.
/OVW	Send	Specify CLOBBER as File Management option.
/TXT	Send	Send file as ASCII with value stored in `@ZMALT`.

Filenames Restrictions

With Xmodem, Ymodem, and Zmodem, you should *not* give a file the same name as a switch since BLAST will assume that the file is a switch and either ignore it (if the switch is unsupported by the current protocol) or look for a file with the name of the folder containing the file (if the switch is supported by the current protocol). In either case, the transfer of the file will not occur and you will get an error message. Filenames (uppercase or lowercase) to avoid are: app, comp=*n*, follow=*nn*, fwd, group, ovw, owner=*nn*, perms=*nnnn*, str, and txt (where *n* is a number from `0` to `9`).

You may work around this restriction by changing your local current or remote current directory to the one containing the file you want to transfer and giving the filename without a path. For interactive sends, change your local working directory by accessing the Local menu and choosing the Chdir command. For interactive gets, change your remote working directory by accessing Terminal mode and giving the “change current directory” command for that operating system.

Alternatively, you may do a scripting workaround. For `SEND`s, change the local working directory by using the `LCHDIR` command. For `GET`s, `TSEND` a “change working directory” command for that operating system. For example, the following script fragment will change the current remote directory on a UNIX machine to /u/sales.

```
TSEND "cd /u/sales", CR
```

Chapter 10

Text Transfers

Introduction

In BLAST session protocol, you may transfer text directly to and from a remote computer using the respective Online commands Upload and Capture.

Uploading Text to a Remote Computer

Uploading is the process of sending text from your system to a remote computer. When you upload, the text being uploaded will display on your screen. The receiving computer does not need to be running BLAST, but it must have a program capable of capturing text and responding to flow control.

Because there is no error detection, characters may be dropped or noise may change the characters in the data stream. The following setup fields, however, can assist in regulating the flow of data during text uploads to help prevent the receiving computer from losing characters: Wait for Echo, Prompt Char, Char Delay, and Line Delay. See Chapter 5 for details on using these functions.

After you have connected, there are three ways to start the upload process with another system:

Manual Method

◊ Select Terminal from the Online menu.

◊ Type the appropriate commands for the remote computer to start a text capture program. On a UNIX system, for example, you might type:

vi remote.fil

which instructs vi to open a new file named remote.fil. You can then use the a command to tell vi to append the uploaded text to remote.fil. Note that an entry is not required in the System Type setup field for this method.

◊ When the remote capture program is ready, press *ATTN ATTN* to exit Terminal mode and then select Upload from the Online menu. Specify the desired local filename, *but not a remote filename.*

◊ After the upload is completed, you will be returned to Terminal mode. Save the file containing the newly captured text, specifying a name if you have not already done so on the command line, and then quit the capture program.

Interactive Automatic Method

Select the Upload command from the Online menu. *You must specify both the local and remote filenames.* Your computer will automatically send the file to the remote system, if text capture is supported by that system.

NOTE: The remote computer type must be entered in the System Type setup field for this method to work because BLAST uses the system.scr library to automate the process. BLAST will start the remote text capture program for you.

BLASTscript Automatic Method

See "Text Transfers" on page 207 for details on scripting uploads.

Downloading Text from a Remote Computer

Downloading is the process of capturing text sent from another system to your computer. When you capture text from a remote computer, the text being downloaded will display on your screen. The sending computer does not need to be running BLAST, but it must have a program capable of sending text and responding to flow control. If flow control is specified in the setup, BLAST will pause transmission for a few moments when the buffers are full. After connecting, there are two ways to start the download process:

Manual Method

◊ Select the Capture command from the Online menu and specify the desired filename for the capture file.

◊ Select Terminal from the Online menu. Type the appropriate command for the remote computer to start typing the text. For example, at the “$” prompt of a UNIX system, you might type:

```
cat test.fil
```

◊ When the download has completed, press *ATTN ATTN* to exit Terminal mode. Turn Capture off by selecting it again.

BLASTscript Automatic Method

See “Text Transfers” on page 207 for details on scripting downloads.

Chapter 11

Secure BLAST

Securing Your System

Securing your system against intrusion is a complex task. "Secure BLAST" is a security tool that provides access for authorized users only. Before discussing the BLAST security utilities in detail, we will examine standard UNIX methods of security.

IMPORTANT: There are many tools for securing a UNIX system, but none of them are foolproof. At best, they will significantly reduce the risk that well-intentioned people will inadvertently access restricted data. These mechanisms will not safeguard your data against a systematic, continuous attempt to "hack" your computer. For more detailed information on system security, please refer to your system documentation or any of the excellent references available concerning UNIX security.

Login

The front line of defense against unwanted intrusion is a properly configured getty or ttymon process running on all dial-in lines. At a minimum, this will force the dial-in user to enter a valid login name and password (or attempt to hack the login process) to gain access. Presuming that the user provides a valid login and password combination, login will start a shell, set the group to which the user belongs, and put the user into his or her home directory. The default shell, group ID, and home directory are all specified in the /etc/passwd file.

To maintain security, each person logging into your UNIX system should have his or her own login and home directory. If logins and home directories are shared, it is impossible to limit directory access only to one user. For more information on setting up logins, refer to your system documentation.

Groups

Each user on the system will belong to one or more groups. Segregating users into groups can help secure your system. For more information on setting up groups, refer to your system documentation and the group, newgrp, and chgrp man pages.

Permissions

The basic operations performed on a file are "read," "write," and, for executable files, "execute." The system can grant or deny access to a file for any of these operations. Read, write, and execute permission can be set for the owner of the file, for users in a particular group, and for all other users on the system.

When you list the files in a directory, the permissions assigned to each file appear as a series of letters. "Read," "write," and "execute" are denoted by the mnemonics "r", "w", and "x." The permissions indicator can contain up to 10 characters, but not every space will necessarily contain a letter. You should think of the permissions indicator as a single initial character plus three groups of three characters. For example, "-rwxr--r--" should be interpreted as:

	Owner	Group	Others
-	{rwx}	{r--}	{r--}

The owner of the file has all three permissions, users in a particular group have only read permission, and all other users have only read permission. If the file is a directory, the initial character will be a “d”. Other specifiers for the initial character and other file permissions exist as well but are beyond the scope of this discussion.

As a general rule, set default permissions for newly created files as restrictively as possible. The tool for setting default permissions is umask, discussed later in this section. If the owner of a file wants to allow expanded access to his files later, he can manually reset the permissions using the chmod command. For more information on changing file permissions, please refer to the chmod man page.

Another parameter that affects permissions is the set-uid bit, which allows a program file to execute with the permissions of its owner rather than the permissions of the user running the program. For more information on the set-uid bit, see your system documentation.

Directories

To the UNIX operating system, a directory is just a file. The same read, write, and execute permissions apply to directories; however, interpretation of execute permission is different. If a directory has execute permission, it is possible to search the directory. As a general rule, a user’s home directory should have read, write, and execute permission for the owner only. This will allow the owner of the directory complete access to his or her files but disallow access to all others.

umask

The umask (user mask) tool is used to establish default permissions when a file is created. If umask has never been set, the operating system will create a set of default permissions, but you should examine them carefully. It is important that you use umask to set default permissions as restrictively as possible while still allowing necessary access to your files. You can use umask to set permissions permanently or to change them for a particular shell session.

Permissions can be denoted both mnemonically and numerically (see table on the next page), where the mnemonic indicates what is permissible, and the numeric indicates what is not permissible.

Permission	Mnemonic	Numeric
Read	r	4
Write	w	2
Execute	x	1

Permissions are restricted according to the numeric values specified in the umask. For example, to deny write access, the umask should have a “2” in it. Numeric values are added together to express the total restriction of permissions set for an owner, group, or others. For example, denial of read, write, and execute permissions is denoted by the number “7”. Conversely, read, write, and execute permission is denoted by a “0”.

To display the default umask type:

umask

The output of umask will be a three-digit number such as:

022

The “0” in the first position indicates that no permissions will be restricted for the owner of the file. The “2” in the second and third positions indicates that write permission will be restricted for both the group and all other users, respectively. This is not a particularly secure umask setting because read and execute permissions are not restricted.

The most restrictive umask is 077, which allows all permissions for the owner but removes read, write, and execute permissions for the group and all other users.

To set your umask for the session to 077, type:

umask 077

To reset your umask permanently, add the above line to your .profile, .cshrc, or other login script. To effect a umask change across the entire system, you can set umask in the /etc/profile file. For more information, refer to your system documentation as well as the umask and profile man pages.

BLAST Protocol File Transfer and Permissions

When a file is transmitted using the BLAST protocol, permissions are set according to the rules described below (see “File Transfer Switches” on page 111 for information on the /PERMS switch and other switches that affect owner and group permissions):

◊ If the file is being transmitted from a UNIX system, and the /PERMS switch is not used, BLAST will attempt to transfer the file with the same permissions as it has on the source machine.

◊ If the file is transmitted using the /PERMS switch, BLAST will attempt to set file permissions on the destination machine according to the permissions specified by the /PERMS switch.

◊ If the file is transferred from a system that does not have a permission structure comparable to the UNIX permission structure, permissions will be set according to the umask equivalent on the receiving system.

Running BLAST from a Restricted Shell

It is possible to set up user accounts with a restricted shell like rsh or rksh. With a restricted shell, a user is unable to edit .profile, change directories, or set the PATH environment variable. Thus, once a user is logged into a restricted shell, he has very limited capabilities.

A restricted shell account normally contains a bin subdirectory and a work subdirectory. The .profile login script will place the user into the work subdirectory. Scripts and executables that the user will be allowed to run should be put into the bin subdirectory.

The system administrator must set the PATH in .profile to point to the user's bin subdirectory. The user's PATH can point to other subdirectories but should not point to /bin. If the user's PATH is set to /bin, the user will be able to start an unrestricted shell and defeat the restrictions imposed by rsh.

Run BLAST from a restricted shell account by setting the PATH and BLASTDIR environment variables to the actual BLAST directory. For example, if BLAST is in the /usr/blast directory and the user's home directory is /usr/jo, add the following to the user's .profile:

```
PATH=$PATH:/usr/blast
BLASTDIR=/usr/blast
SETUPDIR=/usr/jo/work
export PATH BLASTDIR SETUPDIR
```

Alternatively, BLAST can be run from a restricted shell account by creating links to the appropriate files in the user's bin subdirectory and setting the environment variables appropriately. If BLAST is in /usr/blast and the user's home directory is /usr/jo, create the following link:

```
ln -s /usr/blast/blast /usr/jo/bin/blast
```

If the user is going to be running BLAST interactively, you need to make the following links:

```
ln -s /usr/blast/blast.hlp /usr/jo/bin/blast.hlp
ln -s /usr/blast/setgetty /usr/jo/bin/setgetty
ln -s /usr/blast/modems.scr /usr/jo/bin/modems.scr
ln -s /usr/blast/systems.scr /usr/jo/bin/systems.scr
```

The environment variables should be set to:

```
PATH=/usr/jo/bin
BLASTDIR=/usr/jo/bin
SETUPDIR=/usr/jo/work
export PATH BLASTDIR SETUPDIR
```

Limiting Access

Certain procedures are required to ensure data security when using BLAST. These procedures include limiting access to other file transfer protocols and limiting shell access.

Other file transfer protocols do not offer the same data security that the BLAST protocol offers; therefore, you must control access to these other protocols. Common file transfer programs include kermit, ckermit, sx, rx, sz, and rz. If these or any other file transfer programs are on your system, you should segregate dial-in users into groups that do not have execute permission for these programs.

You must also prohibit running the BLAST product in pseudohost mode (host mode using protocols other than BLAST; see "BLAST Operation as a Pseudohost With 10.8x" on page 204).

BLAST can be executed from a restricted shell or in a manner denying a remote user terminal access to the UNIX shell. A shell script that sets the appropriate environment variables and executes BLAST can be run by the user's .profile or other login script.

The following script called go_blast is an example of a shell script that executes BLAST in host mode using the default setup. In this example, the BLAST executable is in the directory /usr/blast/secure and the BLAST support files are in /usr/blast.

```
# go_blast
# A script that sets environment variables and runs BLAST in host mode using
# the default setup
#
PATH=$PATH:/usr/blast/secure         # Append /usr/blast/secure to the
```

```
                                    # current PATH environment variable.
BLASTDIR=/usr/blast                 # Set BLASTDIR environment variable.
SETUPDIR=/usr/blast                 # Set SETUPDIR environment variable.
                                    #
export PATH BLASTDIR SETUPDIR       # Export environment variables to
                                    # subshells--necessary in bourne shell.
blast default -h                    # Run blast in host mode.
exit
#
# End of script.
```

This script should have execute permission and be located in the search path. The following line should be added to the end of the user's .profile or other login script:

```
exec go_blast
```

The exec command substitutes the new process for the calling process. In essence, the process running the .profile or other login script that calls go_blast is transformed into the process running the go_blast shell script. Nothing following the above exec command in the .profile will be executed. Once BLAST finishes running in host mode, the user will be logged off the system.

Using Secure BLAST

Secure BLAST was developed to provide an extra layer of security over existing UNIX restrictions. Secure BLAST not only recognizes UNIX file permissions but can also further restrict access to particular files as well as insuring that a user executes only authorized versions of BLAST on both the local and remote systems. Authorized users can be limited to a very narrow range of available options in transferring files and performing remote operations.

Secure BLAST allows the BLAST administrator to create a database of user passwords, each with individual security options. Authorized users must provide one of these valid passwords in order to gain access to the "secured" version of BLAST. The permissions associated with individual passwords in the database control what files and commands are available to the user. For information about how to transmit user passwords, see "Using the Password" on page 168.

The BLAST administrator can use either the blpasswd or blsecure application provided with BLAST to create and maintain the password database. Whereas blpasswd provides a complete user interface for setup and maintenance of the BLAST password database file, blsecure is a command line utility that is particularly useful when you want to manipulate the password file via a shell or BLAST script.

"Securing" BLAST is a two-step process that consists of creating the password database file and then linking it to a particular BLAST executable file. After creating the password file using either blpasswd or blsecure, another utility, secure, is used to create the link between the password file and the BLAST executable file.

Throughout this chapter the computer running secure BLAST will be referred to as the "host" system, and the computer logging onto the host will be referred to as the "remote" system.

IMPORTANT: Although it is possible to create a password file and link it to a particular BLAST executable file without specifying a full pathname for either file, it is not advisable. You should specify a full pathname for the BLAST executable and password file when using the Secure BLAST utilities. A higher level of security is maintained if neither the password file nor the BLAST file are located in the same directory as blpasswd, blsecure, and secure.

blpasswd

The first step in securing BLAST is to create the password file. The application blpasswd provides a full-screen user interface for setup and maintenance of the password database. The BLAST installation program normally copies blpasswd to the same directory as the BLAST executable. For increased security, blpasswd should be moved to a directory that is accessible only to the BLAST administrator.

Creating and Modifying a Password File

To create a new password database file, execute blpasswd from the command line by typing blpasswd and pressing ENTER. You will be prompted for a filename (Figure 11-1). Type the filename and press ENTER.

FIGURE 11-1

```
                         BLPASSWD Version 2.00
File Name: █
```

Next, you will be prompted to create a master password, which will control future access to the file; type the password and press ENTER. You will then be asked if you want to create the new file (Figure 11-2). Press Y to create the file, or N or C to cancel and exit blpasswd.

FIGURE 11-2

```
                      BLPASSWD Version 2.00

                        Create new newfile

                    Yes  No  Cancel  ( Y/N/C )



_____Password_____  __Permissions___  ______________Comment______________
```

After creating a new password file, blpasswd will display the main screen (Figure 11-3).

To open an existing password file for modification, from the command line type blpasswd followed by a space and the name of the file you want to open. You will then be asked for the master password. Typing the password and pressing ENTER will take you to the main screen (Figure 11-3). If the filename you type does not exist, you will be asked if you want to create the file. Press Y to create the file, or N or C to cancel and exit blpasswd.

FIGURE 11-3

```
                      BLPASSWD Version 2.00

Password file version: 10.7.6        BLAST Serial #
Program file:


     Comment:



_____Password_____  __Permissions___  ______________Comment______________
```

The next step in creating or modifying a password file is to enter data into the file, which consists of two parts: header information and record information. Header information includes master data for the password file; record information includes data in each individual password record.

Header Information

Header information consists of the following master data for the password file: the master password for file edit access; the serial number, name, and location of the BLAST executable to which the file will be secured; and optional comments about the password file.

To enter header information into a newly created file or to edit header information in an existing file, press H from the main screen. You will see a screen similar to the one in Figure 11-4 below:

FIGURE 11-4

```
                          BLPASSWD Version 2.00
                              Modify Header
      Password:
 Serial Number: 0123456789-0-00000
BLAST filename: /usr/local/blast
       Comment: This is a header comment



_____Password_____  __Permissions___  ______________Comment_______________
```

Type the appropriate information into the field highlighted. To move from field to field, press ENTER. After typing data into the Comment field and pressing ENTER, you will be asked if the data you have typed is correct. If it is, press Y; if not, either press C to return to the main screen without saving changes, or press N to move through the fields again for editing. If you do not enter data into any of the fields, pressing ENTER from the Comment field will return you to the main screen. *In order to use a newly created file, you must first fill in the header information fields.*

Following is a detailed description of each field:

Password — user-defined

Specifies the master password, which controls editing access to the database file. You must enter this password in order to edit any part of the database file, either header or record information. The Password field will contain the master password that you entered when creating the file, although it will not be displayed. Press ENTER to retain this password and move to the Serial Number field. If you want to change the master password, type in the new password and press

ENTER. You will be prompted to retype the new password for confirmation.

Serial Number XXXXXXXXXX-X-XXXXX

Specifies the serial number of the BLAST executable that you want to secure on the host system. Type in the 16-digit serial number *with dashes after the 10th and 11th characters* exactly as it appears on the BLAST executable, for example, `0123456789-0-00000`. The serial number and version of BLAST are visible when you press the *HELP* key while running BLAST. *If the serial number of the secured executable and the number in the header information do not match, access will not be allowed.*

BLAST filename user-defined

Specifies the name and path of the BLAST executable file that you are securing on the host system. You must specify the complete path and filename, for example, `/usr/joe/blast` where /usr/joe/ is the directory location and blast is the name of the secured executable.

Comment user-defined

Specifies optional comments regarding the password file.

Record Information

Record information includes the data in each individual password record. This information will determine who is allowed access to the secured version of BLAST and what permissions that user will have. Record information includes: a user password for access to the secured version of BLAST; the permissions associated with that password; the serial number of the remote BLAST executable associated with that password; the directory where files will be transferred; masks to control what files can be transferred; and optional comments about the record.

Adding, selecting, and editing records are all controlled by the following set of command keys issued from the main screen:

A Add a new record. All edit fields will be blank.

T Select the top (first) record.

D Move down one record.

U Move up one record.

B Select the bottom (final) record.

F Find a record by password and select it; blpasswd will prompt you to enter the password.

E Edit an existing record (also accessed by selecting a record and pressing ENTER).

H Edit the header information.

Z Zap a record (mark it for deletion). The record will be marked as unused but not physically removed. *This command has the effect of disabling that record and its password.* When a record is "zapped," a "z" is displayed after the permissions.

Zapped passwords cannot be used for a new record until they have been "reclaimed" (see R). When the zap command is used on an already zapped record, the record is "unzapped" and the password and record are enabled once again.

R Reclaim zapped password for possible reuse and delete zapped record. Record numbers may change after use of the reclaim command.

Q Quit blpasswd. You will be prompted to save any changes.

After you have designated a file for creation or editing using the command keys and pressed ENTER, blpasswd will display a screen similar to the one shown in Figure 11-5 below:

FIGURE 11-5

```
                              BLPASSWD Version 2.00
                                  Modify Record
        Password: newuser
     Permissions: GSTLERPCOA
   Serial Number: 9876543210-0-00000
  Home Directory: usr/jon
    Include mask: *.*
    Exclude mask: *.exe
         Comment: user with full permissions


  _____Password_____  __Permissions___  ______________Comment______________
=>newuser             GSTLERPCOA        user with full permissions
  olduser             G  L RP           user with limited access
```

Type the appropriate information into the field highlighted. To move from field to field, press ENTER. After typing data into the Comment field and pressing ENTER, you will be asked if the data you have typed is correct. If it is, press Y; if not, either press C to return to the main screen without saving changes, or press N to move through the fields again for editing. If you make no changes in any of the edit fields, pressing ENTER from the Comment field will return you to the main screen.

Following is a detailed description of each field:

Password user-defined

Specifies the user password for an individual record. This field is blank only for new records. A password cannot be altered once it has been saved in the database, but it can be deleted and made available for reuse with a new record by applying the reclaim command to a "zapped" file (see R command on preceding page).

Permissions GSTLERPCOA or M

Specifies the permissions allowed by the user during a BLAST session on the host system. Type in the letter or letters that specify the permission(s) allowed (see list below).

IMPORTANT: BLAST does not override standard UNIX permissions. For example, even though a user may have BLAST permission to rename a file, he cannot do so if he does not have UNIX write permission for that file. Likewise, he cannot change directories if he does not have UNIX permission to do so.

The following permissions are available:

- A Append – User can append to a file.
- C Change directory – User can change directories.
- E Delete – User can delete a file.
- G Get – User can get a file.
- L List – User can list directory contents.
- M Master – User can perform all available operations.
- O Overwrite – User can overwrite a file.
- P Print – User can print a file.
- R Rename – User can rename a file.
- S Send – User can send a file.
- T Type – User can type a file.

Serial Number XXXXXXXXXX-X-XXXXX

Specifies the serial number of the BLAST program that the user is executing on the remote system. Type in the 16-digit serial number *with dashes after the 10th and 11th characters* exactly as it appears on the remote BLAST executable, such as `0123456789-0-00000`. The serial number and version of BLAST are visible when you press the *HELP* key while running BLAST.

This field may contain the wildcards "?" and "*" to match single or multiple numbers, respectively. For example, `0123456789-?-*` will accept any serial number that begins with 0123456789, has any number as the eleventh digit, and has any combination of numbers for the last five digits.

If the serial number of the remote executable and the serial number in the record do not match, access will not be allowed.

Home Directory user-defined

Specifies the directory to which the host computer changes upon validating the password. Files will be transferred into and out of this directory unless the user has permission to change to another directory. The user must have normal UNIX permission for the directory named as the home directory or he will see the error message "Invalid Home directory!" when sending his password. *If this field is left blank, access will not be allowed.*

Include mask user-defined

Specifies the files that can be accessed by the remote user. If a directory is specified in the include mask, the user must have both BLAST chdir permission and UNIX permission for that directory. The wildcards "?" and "*" may be included anywhere in the include mask. For example `file?.dat` would allow a user to transfer file1.dat and file2.dat, while `*.dbf` would allow access to all .dbf files in any directory. *If this field is left blank, no files can be accessed.*

IMPORTANT: The include mask will *never* override your operating system's permission or access system. The user will not be able to access a file or directory using BLAST unless UNIX read, write, and execute permissions are correctly set.

Exclude mask user-defined

Specifies files to be excluded from the include mask. For example, if the include mask is set to `file*.*` and the exclude mask is set to `*.c`, a file named file34.a would pass through, but a file named file34.c would not be accessible to the remote user. If this field is left blank, *all* files matching the include mask will be accessible.

IMPORTANT: The exclude mask *will* override your operating system's permission or access system. Even if UNIX read, write, and execute permissions normally allow access to a file or directory, BLAST will deny access if it matches the exclude mask.

Comment user-defined

Specifies optional comments regarding each record.

blsecure

The application blsecure is a command line utility that, like blpasswd, sets up and maintains passwords and permissions for BLAST users. Unlike blpasswd, it does not have an interface and does not require interactive input from the BLAST administrator, thereby making it ideal for use from a shell script or a BLAST script.

blsecure Command Line Parameters

The BLAST installation program normally copies blsecure to the same directory as the BLAST executable. For increased security, it should be moved to a directory that is accessible only to the BLAST administrator. To run blsecure, use the following format:

blsecure *passwordfile masterpassword* { c | h | g | f | p | a | z | r } [*options*]

where *passwordfile* is the filename of the password file; *masterpassword* is the master password that grants editing access to the password file; c, h, g, f, p, a, z, and r are parameters that allow the user to create, search for, and modify password files; and *options* are arguments of the single-letter parameter.

The single-letter parameters and their accompanying arguments are described in detail below. Note that only one single-letter parameter and its arguments can be used on a command line.

c *sn blastexe* [*comment*]

creates a new password file with the filename and master password specified on the command line.

sn	Serial number of the host BLAST executable.
blastexe	Full path and name of the host BLAST executable.
comment	Optional comment.

h [*newmast sn blastexe* [*comment*]]

allows you to modify header information. You can change one or more of the data fields with one of the arguments listed below; however, all of the arguments except the *comment* must be included on the command line. Any argument that you want to remain unchanged must be typed on the command line exactly as it is in the existing header.

newmast	New master password for the password file.
sn	New serial number of the host BLAST executable.
blastexe	New path and name of the host BLAST executable.
comment	New comments.

If you do not specify information for each argument (except comment), blsecure will return an error. If no comment currently exists in the password file header, you can leave out this argument or specify a new comment. If a comment does exist in the header, you can replace it with this argument or, if you leave the argument blank, blsecure will delete the currently existing comment.

When specifying the h parameter alone, as in blsecure *passwordfile masterpassword* h, the header information for that password file will be displayed as shown in Figure 11-6 below:

FIGURE 11-6

TEST3	(master password)
10.7.6	(password file version)
0123456789-0-00000	(host BLAST serial number)
newfile	(password filename)
usr/local/blast	(host BLAST filename)
This is a new comment	(comment)

a *pwd perm sn home inc exc* [*comment*]

adds a new record. You must have an entry for each argument (see description of the h parameter). This must be a new entry with a new password. See “blpasswd” on page 156 for complete descriptions of the data fields associated with the following arguments.

pwd	Password of the existing record.
perm	Permission specifier as described below.
sn	Serial number of the remote BLAST executable.
home	Directory to which the host computer changes upon linking in Filetransfer mode.
inc	Include mask for specifying files that may be obtained by the remote user.
exc	Exclude mask for screening files that pass the include mask. If you do not want an exclude mask, substitute double quotation marks (" ") on the command line.
comment	Optional comment.

The following are the possible hexidecimal values for the perm argument; they can be added together to form the total permission value:

0001	User can get a file.
0002	User can send a file.
0004	User can type a file.
0008	User can list directory contents.
0010	User can delete a file.
0020	User can rename a file.
0040	User can print a file.
0080	User can change directories.
0100	User can overwrite a file.
0200	User can append to a file.
7FFF	User can use all functions.

For example, if you wanted to create a new record with the data illustrated in Figures 11-6 (preceding page) and 11-7 (next page), you would type the following:

```
blsecure newfile TEST3 a site23 0001 1234567891-0-00000 \
/usr/sites *23.dat mast*.* Password for site23
```

p *pwd perm sn home inc exc* [*comment*]

puts information into an existing record. The arguments are identical to the a command above.

g *recnum*

searches a record by its number and displays the record as shown in Figure 11-7 below.

recnum	Record number for a particular entry. The record number of the first entry is zero.

NOTE: After reclaiming a password for possible reuse (see r), records may have different numbers.

f *pwd*

searches for a record by its password and displays the record as shown in Figure 11–7 below.

pwd	Password for the individual record.

FIGURE 11-7

site23	(password)
0001	(permission in hexadecimal)
1234567891-0-00000	(remote serial number)
/usr/sites	(home directory on host)
*23.dat	(include file mask)
mast*.*	(exclude file mask)
Password for site 23	(comment)

z *pwd*

marks a record for deletion but does not physically remove it. A zapped record can be "un-zapped" and reactivated if it is the target

of a subsequent z command. A zapped password cannot be reused until it is reclaimed and the zapped file deleted (see r).

pwd	Password of the record to be deleted.

r

reclaims a “zapped” password for possible reuse and deletes the zapped record. Record numbers, used by the g command, may be reordered by using r.

blsecure Error Codes

Errors that occur while running blsecure are due to physical causes, such as the file not being found, or read and data errors, such as g failing to locate a specified record. To help prevent unauthorized access to the password database, returned error codes do not indicate anything other than a general failure.

secure

After you create a password file, use secure to establish a link between the BLAST executable and the password file. In order to use this utility, the BLAST executable must exist with “write” privileges for the administrator. secure should be made accessible only to the BLAST administrator by means of operating system permissions or privileges.

Running secure

The BLAST installation program normally copies secure to the same directory as the BLAST executable. For increased security, secure should be moved to a directory that is only accessible to the BLAST administrator.

Execute secure from the command line by using the -s switch in the following format:

secure *blastexecutable* -s *passwordfile*

where *blastexecutable* specifies the complete path and filename of the BLAST executable, and *passwordfile* specifies the complete path and name of the password file you wish to secure, as in the following example:

secure /local/blast -s /private/password.fil

In this example, /local/blast is the pathname for the BLAST executable contained in the directory /local, and password.fil is the name of the password file located in the /private directory. The -s switch links the password file to the executable.

After BLAST is secured, you can determine the name of the password file attached to it by using the -d switch in the following syntax:

secure *blastexecutable* -d

For example,

secure /local/blast -d

will respond with a message similar to the following:

Secure - Version 2.0
private/password.fil (/local/blast)

Using the Password

After you have created a password file and secured your host system, a remote user must use one of the passwords in the password file in order to access the host through the BLAST Session protocol. The password is transmitted from the remote to the host system by the same method used for transmitting a transfer password in the BLAST Session protocol. Note, however, that the secure password *supersedes* the transfer password; therefore, the remote user will only be prompted for the secure password even though a particular setup may also contain a transfer password.

The password is sent from the remote system with the Send command of the Filetransfer menu or with a BLASTscript `FILETRANSFER` statement. If a Send command is issued, the following special format for the local filename must be used,

`!password=`*`your_password`*

where *`your_password`* represents one of the passwords stored in the database file on the host system. The remote filename field is left blank as are the text, overwrite, and append options. If the correct password is successfully sent, the remote user will see a message stating that the password has been validated. The user must type the password exactly as it appears in the password record, and the serial number of BLAST being executed must match the serial number in the password record.

In a BLAST script, the same special local filename format must be sent to the host computer. For example:

```
FILETRANSFER
SEND
!password=blue2


SEND
myfile.rpt
yourfile.rpt
ta
ESC
```

Since no remote filename or Send options are used, two blank lines follow the password line. See “File Transfers with BLAST Session Protocol” on page 194 for information on scripting file transfers.

Chapter 12

Introduction To Scripting

Starting Out

Scripts allow BLAST to automate communications tasks. Scripts are often used for tasks such as logging into remote hosts and handling the details of communications sessions that are repetitive or that inexperienced users would find overwhelming. This chapter introduces the BLASTscript language and describes an important feature of BLAST that aids scripting—Learn mode. With Learn, BLAST writes your scripts so that learning scripting is made easier.

Executing BLAST Scripts

BLAST scripts can be invoked using one of three different methods.

◊ From the Online menu, select the Script command. When prompted for the script name, enter the name of the file. This interactive method of starting a script is preferable when you wish to automate only a portion of your communications session.

◊ In a setup, enter the name of a BLAST script in the Script File field. After the setup is loaded into memory and the Online command is selected from the Offline menu, the script named in the setup will execute automatically. This is useful if you always use a specific script with a particular setup.

◊ From the operating system command line, specify a BLAST script name with the -s switch (see “Command Line Switches” on page 10). The script specified on the command line takes precedence over a script listed in the setup Script File field.

You can include a directory path when you specify a script filename. If you do not name a directory, BLAST will first search the current directory and then the SETUPDIR directory.

To abort a script completely, press *ATTN ATTN*. To abort a script after the currently executing statement completes execution, press *ATTN* once.

Writing a Script

The best way to learn how to write a script is by doing it. First, start a word processing program or a text editor on your computer. If you prefer to use a word processor for creating script files, be aware that your scripts must always be saved as text files, not word processor documents. Your scripts should be saved in the directory from which you will execute BLAST, or in the SETUPDIR directory. These are the only two locations in which BLAST searches for script files if you have not specified a search path.

After starting the editor, type in the following short script:

```
# hello.scr
#
# Just wanted to say hi
#
.begin
  display "Hello, world!"
  return
#
# End of script.
```

Save this file under the name “hello.scr” and go to BLAST’s Online menu. Choose the Script option and enter the filename

`hello` ENTER

When hello.scr executes, it displays the message

Hello, world!

on your screen and then returns control to you.

About hello.scr

As simple as hello.scr is, it illustrates several important scripting concepts. All the lines starting with "#" are comments explaining the functions of the script commands and are not displayed. You may be surprised how quickly you can forget why you wrote a particular script or how an especially difficult section of code actually works. Comments can clarify what you are trying to accomplish with your script.

In hello.scr, the line beginning with a period, `.begin`, is called a label. A label serves not only as a supplemental comment but also as a destination for the script to go to in a `GOTO` command, discussed later. Labels can be eight characters in length, not counting the initial period.

The `DISPLAY` command causes text to be displayed on your local computer screen; it does not cause text to be transmitted through the serial port. Another script command, introduced later, performs this task.

Finally, the `RETURN` command returns control of BLAST to you.

A Sample Script

To learn more about scripting, it is helpful to imagine a problem that can be solved through scripting. For instance, suppose a medical office needs to call an insurance company each evening to file insurance claims on behalf of patients who have visited the doctor that day. Pam, the system administrator for the medical office, collects the claims into a single file called pt_claims. Since the insurance company also uses BLAST software for data communications, Pam will use the BLAST Session protocol to transfer pt_claims to the insurance company. The company has determined that Pam's daily claims file should be given the name logan56021.dat on the insurance company system. Therefore, Pam wants a script to perform the following tasks:

1. Connect to the remote system.

2. Send the claims file as a text file.

3. Disconnect.

A script that meets these requirements is illustrated below. The script dailyrpt.scr is certainly more complicated than hello.scr, but the same sections that were originally outlined are present. To make it easier to discuss the script, we will refer to the line numbers shown in brackets next to the script statements. *You would not include these numbers in an actual script.*

```
[ 1] # dailyrpt.scr
[ 2] #
[ 3] # A script to send daily medical reports to
[ 4] # the insurance company
[ 5] #
[ 6] # Section 1: CONNECTING
[ 7] #
[ 8] .begin
[ 9]   set @ONERROR = "CONTINUE"
[10]   connect
[11]   if @STATUS = "0" goto .xfer
[12]   display "No Connection! Error code: ", @STATUS
[13]   return
[14] #
[15] # Section 2: TRANSFERRING
[16] #
[17] .xfer
[18]   filetransfer               # enter BLAST protocol
[19]   send                       # prepare to send a file
[20]   /usr/accounts/pt_claims    # local filename
[21]   logan56021.dat             # remote filename
[22]   t                          # specify text file
[23]   esc                        # exit Filetransfer mode
[24]   if @EFERROR not = "0"
[25]     display "An error occurred during file transfer."
[26]     display "Please examine the log file."
[27]   end
[28] #
[29] # Section 3: DISCONNECTING
[30] #
[31] .finish
[32]   disconnect
[33]   return
[34] #
[35] # End of script.
```

CONNECTING (Section 1)

The first section of the script (`.begin`) establishes the connection with the insurance company. Line 9 sets a variable called `@ONERROR`. In a BLAST script, all variables begin with "`@`". Some variables are reserved, meaning that they are defined by BLAST for special purposes; other variables can be created by you (see *BLASTscript Reserved Variables* on page 263). `@ONERROR` is a reserved variable that determines how BLAST will respond to routine (nonfatal) errors. By giving `@ONERROR` the value `CONTINUE`, Pam is telling BLAST to skip error messages rather than pause and wait for a human operator to respond.

Line 10, the `CONNECT` statement, is responsible for a great deal of work. The `CONNECT` statement, like Connect from BLAST's interactive menus, initializes the modem, dials the insurance company, and logs into the company system. All of this information—the modem type, phone number, remote system type, and account information—is taken from the setup (see *Connecting and Disconnecting* on page 211).

Line 11 demonstrates how scripts are programmed to make choices with the `IF` (conditional) statement. After the `CONNECT` command executes, it sets the value of `@STATUS` to indicate whether or not the connection was successful. The `IF` statement tests the value of `@STATUS` in its conditional clause. If `@STATUS` equals `0`, the connection was successful and the script performs the `GOTO` command, sending the script to the section labeled `.xfer`, which controls file transfer.

```
if @STATUS = "0" goto .xfer
```

conditional statement — *executes if conditional clause is true*

If `@STATUS` equals any value other than `0`, script execution continues on line 12, displaying "No Connection" and an error code. At this point, `RETURN` aborts further execution of the script and control is returned to the user.

TRANSFERRING (Section 2)

The second section, under the `.xfer` label, begins with the `FILETRANSFER` statement. The `FILETRANSFER` statement works like the Filetransfer command of the Online menu. When it is executed, BLAST attempts to start the BLAST software on the remote computer, and the script pauses until Filetransfer mode is entered or a time limit expires. The exact events that occur when the `FILETRANSFER` command is executed depend on the setting of the System Type setup field (page 67).

The next four lines (19–22) provide the information BLAST protocol needs to send the required file as a text file. If another protocol were used, this section would be scripted differently (for more information on scripting for alternative protocols, see Chapter 13). Line 23, ESC, ends the filetransfer session.

Lines 24–27 illustrate another form of the IF command, IF-END. With IF-END, several lines of script can be executed in a block if the conditional clause is true. In line 24, the @EFERROR reserved variable is tested, which indicates if any errors occurred during a BLAST protocol file transfer. If @EFERROR equals 0, no errors were encountered. For any value other than 0, two messages (lines 25–26) are displayed and the IF statement ends. In either case, the script advances to the .finish label.

DISCONNECTING (Section 3)

The final section of the script, under the .finish label, begins with the DISCONNECT command. Like CONNECT and FILETRANSFER, DISCONNECT performs the same operation as the corresponding command of the Online menu. As you become more familiar with BLAST's scripting language, you will discover that many script commands are similar to the options on BLAST's interactive menus. RETURN ends the script and returns control of BLAST to you.

Learn Mode

An important aid to writing your own scripts is BLAST's Learn mode. With Learn, you perform a communications task exactly as the script should perform it, and BLAST creates the script from the actions you take. Typically, the Learn script serves as a "rough draft" of the final script. To start Learn mode, select Learn from the Offline menu. BLAST prompts you to name the Learn script. Note that Learn mode does not function with PASSTHRU.

Suppose that you wanted to write a script to log into a computer for which there is no standard system type in the BLAST setup. A bank's computerized account service, for example, may have an unusual login. Assume that after the modems connect, the bank issues the prompt "MIDAS>," waits for your user identification (AlbertyArtCo), and then issues the prompt "?:".

To help you write your login script, start Learn mode and then proceed to log in as usual, being careful to avoid spelling errors and other trivial mistakes. When you finish, return to the Offline menu and select Learn again to turn off Learn mode.

The following is an example of what the Learn script might look like:

```
# BLAST Learn mode script
# Original filename: bank.scr
# Date:  09/1/95
# Time:  11:00:00
#
CONNECT
# entering TERMINAL mode
#
ttrap 6, "\012\015MIDAS>"
tsend "Alber"
tsend "tyArtCo", CR
ttrap 3, "\012\015\012\015\?:"
# exiting TERMINAL mode
# RETURN  commented out for appending
```

Even though the script has a strange appearance, you can decipher it. TSEND is the script command for transmitting text through the serial port. This command is used for sending the user ID to the bank. TTRAP is used for checking text coming into the serial port, so it is used for detecting the prompts issued by the bank's system. Without doing any more work, this script will actually perform the login.

Editing the Learn Script

Because BLAST cannot distinguish the meaning of any of the data entering or leaving the serial port, Learn mode may "break" strings of text inappropriately. Editing the Learn script to make the TSEND statements meaningful to human readers is a good idea, but it is not necessary. Likewise, TTRAP statements may contain unneeded characters when scripted by Learn mode. In the example above, \012 is the octal representation of the line feed and \015 is the octal form of the return character. These characters are not needed to detect the prompts issued by the bank, so they may be edited for clarity.

After your have cleaned up the Learn script, it could look like this:

```
# bank.scr
#
# A script to log into the bank
#
.begin
  CONNECT
  ttrap 6, "MIDAS>"
  tsend "AlbertyArtCo", CR
```

```
  ttrap 3, "?:"
  return
#
# End of script.
```

Now the script can be read more easily. After connecting, the script will wait for up to six seconds for the string “MIDAS>.” Next, the script sends the string “AlbertyArtCo” and a carriage return. Finally, the script waits for up to three seconds for the “?:” prompt and then returns control to you.

Polishing the Learn Script

After being edited, the Learn script makes better sense to human readers, but it can still be improved. Take a moment to assess it. What’s left to be done?

One area for improvement is in error handling. You saw earlier that `@STATUS` could be tested after the `CONNECT` command to determine whether a connection was established. Similar error checking should be added to the Learn script.

Another area for improvement is in the use of variables. At present, the user ID is “hard-coded” into the script, meaning that it has a fixed value. If the userid is placed in the appropriate field of the setup, the script can access it with the `@USERID` reserved variable. Thus, a more polished version of the Learn script might look like:

```
# bank.scr
#
# A script to log into the bank
#
.begin
  CONNECT
  if @STATUS not = "0" return
  ttrap 6, "MIDAS>"
  tsend @USERID, CR
  ttrap 3, "?:"
  return
#
# End of script.
```

As you can see, Learn mode and your own knowledge of BLAST’s scripting language simplify the process of automating your communications tasks.

Writing Your Own Scripts

You have now seen enough of the scripting language to begin writing your own scripts. You may wish to read Chapter 13, which describes techniques for working with disk files, manipulating strings, and interacting with programs in your system. Chapter 14 discusses the BLAST method of connecting and disconnecting, which relies heavily on scripts. Chapters 15 and 16 serve as reference guides for all scripting commands and reserved variables. Many examples are included in these chapters to help you get started. In addition, sample scripts are available for download from Blaster (see "Connecting to Blaster" on page 47).

Chapter 13

BLASTscript Topics

Scripting Basics

Although scripts can address a wide range of communications needs, most scripts handle a limited number of common tasks, such as capturing text to a file, displaying information on the screen, and communicating with other programs in the computer. In this chapter we will demonstrate scripting techniques for such tasks.

Programming Style

It may sound strange to say that a script should conform to a certain "style," but following a logical style will make it easier for others to understand your script. For example, indenting sections of script that execute together, such as the code in a conditional (`IF-END`) block, is a simple stylistic convention that helps readability, as in the following script:

```
# Start of script
#
.begin
  display "Hello, world!"
  if @EMULATE = "TTY"
    display "Your emulation is set correctly"
  end
  else
    set @EMULATE = "TTY"
    display "Your emulation is now TTY"
  end
  return
#
# End of script
```

Your programming style also affects how efficiently the script will execute. BLAST scripts are interpreted, meaning that BLAST deciphers the instructions in each line of your script as it executes. To make your script run most efficiently, you should:

◊ Use spaces between expressions. For instance, the script interpreter can evaluate the first line in the example below more easily than it can the second line because of the spaces placed around "=".

```
if @STATUS = "0" set @mystat = "GO"

if @STATUS="0" set @mystat="GO"
```

◊ If certain labels in your script will be frequent destinations for the `GOTO` command, place those labels near the beginning of the script. BLAST looks for labels from the start of the script and works down.

Legal and Illegal Expressions

An error that you may encounter during script development is "illegal menu selection." This error indicates that BLAST has encountered a command in your script that it could not execute. Every line in a script must be executable or contain a comment preceded by #. Blank lines are almost never executable (except for special cases discussed later); thus, do not use blank lines in a script to separate lines of code visually. If BLAST encounters a blank line in a script where it is unexpected, the script interpreter will generate the "illegal menu selection" error.

ILLEGAL	LEGAL
`if @STATUS = "0"`	`if @STATUS = "0"`
	`#`
`disconnect`	`disconnect`
	`#`
`end`	`end`
`return`	`return`

A typing mistake in a script line can also generate an error message. For example, a line such as

```
ig @STATUS = "0"
```

will generate the “illegal menu selection” error because “ig” is not a valid script command.

The Status of @STATUS

The result of many script operations is reported in the reserved variable @STATUS, which has a number of functions, including indicating whether an error occurred during the CONNECT command and identifying which item in a list of target strings was detected by TTRAP. Because @STATUS is affected by so many script operations, you may need to save the value of @STATUS in a “safe” variable so that you can refer to it later in your script, as in the following example:

```
# Following is the target list:
#
  ttrap 5, "Apples", "Oranges", "Peaches"
#
# Save @STATUS in a user-defined variable.
#
  set @fruit = @STATUS
#
# @STATUS will be changed below by the DISCONNECT statement
#
  disconnect
  if @STATUS = "0" display "Disconnected OK"
  else display "Disconnect failure!"
  if @fruit = "0" display "No fruit was selected"
  if @fruit = "1" display "Apples are delicious"
  if @fruit = "2" display "Oranges are tasty"
  if @fruit = "3" display "Peaches are nice, too"
  return
#
# End
```

For a list of all the commands that set @STATUS, see “Commands That Set @STATUS” on page 223.

The CALL Command

When you set out to write a complicated script, ask yourself whether the script is made up of logically distinct sections. If so, you may be able to code each section as a separate script and write a “master” script that calls each section as required, checking for errors. Working with several small scripts is generally preferable to a single large one because it is easier to follow the logic of the program and find errors. The CALL command is used to transfer execution to another script, for example

```
call "getdata"
```

calls the script named “getdata.” When the RETURN command is executed in the called script, control returns to the calling script:

```
return [exit_code]
```

The exit code is optional. When control is returned to the calling script, the value of @STATUS in the calling script will be equal to the value of the exit code. For example, the script testone.scr would call the script testtwo.scr as follows:

```
# testone.scr
#
  display "This script calls testtwo.scr”
  call "testtwo.scr"
#           ...
```

At this point, testtwo.scr executes:

```
# testtwo.scr
#
  ask "Enter a number: ", @input
  return @input
#
# End
```

The value of @STATUS in testone.scr has now been set to the value of @input entered in testtwo.scr, and testone.scr continues with the remainder of its commands:

```
# ...
  display "Now @STATUS = ", @STATUS
  return
#
# End
```

A script that has been called may call another script, a process known as “nesting.” Scripts may be called recursively to the limit of available system resources.

All variables in a script are global, meaning that they can be read and changed anywhere. For example, you can write a script that only sets the variables you will use. Your “master” script then calls this script at the beginning of execution. The master script and any other scripts you call afterward will “see” the variables that you created.

Executing in a Loop

To create a loop, you can write a script to keep track of a loop counter and use the `GOTO` command:

```
# looping demo number 1
#
  set @count = "10"
.loop
  display "Countdown: ", @count
  let @count = @count - "1"
  if @count not = "0" goto .loop
  display "BLAST off!"
  return
```

Running the script would result in the following display on your screen:

Countdown: 10
Countdown: 9
Countdown: 8
Countdown: 7
Countdown: 6
Countdown: 5
Countdown: 4
Countdown: 3
Countdown: 2
Countdown: 1
BLAST off!

An alternative method of looping uses the REPS command. With REPS, the previous script could be written as:

```
# looping demo number 2
#
  reps 10
.loop
  display "Counting down..."
  if reps goto .loop
  display "BLAST off!"
  return
```

Since testing the value of REPS in an IF statement automatically decrements it, REPS is a more compact way of executing a loop than a loop counter. In the example above, the GOTO statement is executed while REPS is greater than zero, so that the loop is exited after the message "Counting down..." has been displayed 10 times. As shown in the illustration below, this method of writing the script produces a different display than that of a loop counter. Note that if the number of repetitions is taken from a variable, the countdown occurs, but the variable retains its initial value.

Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
Counting down...
BLAST off!

Manipulating Text

A number of script commands are available for manipulating text files and text strings. The commands that work with text strings include:

`STRCAT` *`string1`*`,` *`string2`*`, [, ...]` – Combine two or more strings to make a single, longer string. The longer string replaces *`string1`*.

`STRINX` *`string1, string2`* – Find the first occurrence of *`string2`* in *`string1`*. `@STATUS` holds the position of the first character in *`string1`* where a match was found.

`STRLEN` *`string1`* – Find the length of a string. `@STATUS` is set to the value of the length.

`STRRINX` *`string1, string2`* – Find the last occurrence of *`string2`* in *`string1`*. `@STATUS` holds the starting character position of the last occurrence in *`string1`* where a match was found.

`STRTRIM,` *`string1, position1, position2`* – Extract a substring of *`string1`* beginning at *`position1`* and ending at *`position2`*. After the substring has been extracted, the value of *`string1`* is set to substring.

There are other commands for string manipulation, such as the commands to find the ASCII value of a character, to convert all characters in a string to upper or lower case, and to request interactive string input from the user. These and other commands for string manipulation are discussed in Chapter 15.

The following example illustrates the use of string commands:

```
# String demo - extract first and last name from a string
#
# Set variables
#
  set @name = "Johnson, Alfred"
  set @first = @name
  set @last = @name
#
# Find the comma in the name string
#
  strinx @name, ","
#
# Move to last char of last name and extract last name
#
  let @STATUS = @STATUS - "1"
  strtrim @last, 1, @STATUS
  display "Client's last name: ", @last
#
# Move forward to first char of first name and extract
# everything from there to the end of the string
#
  let @STATUS = @STATUS + "2"
  strtrim @first, @STATUS
```

```
  display "Client's first name: ", @first
#
# Rebuild full name by concatenating first and last names
#
  strcat @first, " ", @last
  display "Client's full name: ", @first
  return
#
# End of script.
```

Capturing Text

Two commands, TCAPTURE and SETTRAP, are available for capturing text as it enters the serial port. The TCAPTURE command is used if the text is to be placed in a disk file. The following script illustrates a simple implementation of TCAPTURE.

```
# Capture demo
#
  tcapture on "sales.rpt"
#
# Pause script until 4 sec of "quiet" elapses
#
  wait 4 idle
  tcapture off
#
# End of script.
```

The TCAPTURE command itself does not initiate the text capture. Text capture starts when a WAIT, TSEND, TTRAP, or TUPLOAD command is executed.

The second method, SETTRAP, allows incoming text to be captured into a script variable. The SETTRAP command itself does not cause any text to be captured, but it prepares TTRAP to capture text by setting a variable into which the captured text is to be saved and specifying a limit on the number of characters saved into the variable. A simple form of SETTRAP/TTRAP is:

```
# Settrap/ttrap demo
#
settrap @input, 65   # Capture up to 65 char till end of
                     # line reached
ttrap 30, "^M^J"
#
# End of script.
```

In this example, up to 65 characters are saved into the variable @INPUT. The string ^M^J (carriage return/line feed) triggers the end of the captured text, which includes the trigger string and any text preceding the trigger—up to 65 characters. If no incoming characters match the trigger within 30 seconds, the last 65 characters of text are saved to the variable @INPUT.

More complex forms of the TCAPTURE and SETTRAP commands are described in Chapter 15.

Reading and Writing Text Files

A script can read and write entire lines of text from a text file. As many files can be open at a time as there are file handles available in your system. The commands for opening a file are:

FOPENA *handle, filename* – Open a file for appending.

FOPENR *handle, filename* – Open a file for reading.

FOPENW *handle, filename* – Open a new file for writing (deletes existing file).

These commands must specify two pieces of information: the filename and a file handle. The file handle is an integer that other commands in the script will use to refer to the file. @STATUS is set to the value 0 if the file is opened successfully.

The commands for reading, writing, and closing files are:

FREAD *handle, variable* – Read a line of text.

FWRITE *handle, string* [, *string*] – Write a line of text.

FCLOSE *handle* – Close the file.

To be read properly, a line of text cannot be longer than the maximum length of a variable, which is

10.7x	139 characters.
10.8x	1,024 characters.

When read and write operations are successful, @STATUS is set to 0. If they are unsuccessful—for example, a script attempts to read past the end of a file—@STATUS is set to a nonzero value.

Following is an example of a script that uses the file handling commands:

```
# File read/write demo
#
# Open modems.scr and count the number of lines.
# Write the result in a new file called line.cnt.
#
.begin
  clear
  set @file = "modems.scr"
  fopenr 1, @file
  if @STATUS not = "0"
    werror "Can't open modems.scr"
    return
  end
  fopenw 2, "line.cnt"
  set @count = "0"
  display "One moment, please."
  cursor 10, 6
  put "Reading line #"
.loop
  fread 1, @input
  #
  # If @STATUS is 0, count line and return for another
  #
  if @STATUS = "0"
    let @count = @count + "1"
    cursor 10, 21
    put @count
    goto .loop
  end
.continue         # end of file!
  fwrite 2, @count, " lines in modems.scr."
  fclose 1
  fclose 2
  display "Done! Check line.cnt for line count."
  return
# End of script.
```

Thoughtful screen displays help users gain a sense of being "in good hands." Informing users of the progress of a lengthy job, such as a file transfer, frees them to do other things while the software does its job. Displaying too much text onto the screen at once or neglecting the screen completely, however, can make users wonder instead if their session has malfunctioned. BLAST's scripting language provides a number of commands and reserved variables for controlling the screen to present the right amount of information.

Turning Off the Screen

For some applications, you may wish to turn off regions of the screen while running a script. (To disable screen displays altogether, include the `-n` switch on the command line when you start BLAST; see "Command Line Switches" on page 10) The following reserved variables control particular regions of the display:

`@USERIF` – The user interface area, or menu area, at the top of the screen.

`@SCRLREG` – The scrolling region in the middle of the screen.

`@TRANSTAT` – The File Transfer Status Area of the screen.

Set these variables to `0` or `OFF` to disable the corresponding screen areas. Set the variables to `1` or `ON` to enable them. For example, if you do not want the BLAST menus to be displayed while your script is running, you would put the statement

```
set @USERIF = "0"
```

in your script. The top four lines of the display would then become part of the scrolling region. *You must remember to turn the menu region back "ON" in the script or the user will NOT be able to see the BLAST menus after the script is finished.*

Displaying Text in the Menu Region

Two script commands permit you to display text in the menu region:

`WRITE` *`string`* `[,` *`string`*`]` – Prints a message.

`WERROR` *`string`* `[,` *`string`*`]` – Prints a message in the menu region and then waits for the user to press a key. (The script will not pause if `@ONERROR` is set to `CONTINUE`.)

These commands are normally used for displaying errors or progress messages.

Displaying Text in the Scrolling Region

The most common way to display text in the scrolling region is with the `DISPLAY` statement described on page 228. The `DISPLAY` command prints a string or a list of strings at the current cursor position; depending on the emulation you have chosen, the cursor may or may not advance to the next display line.

Another method of displaying text uses a pair of commands, `CURSOR` and `PUT`:

`CURSOR` *`row, column`* – Position cursor.

`PUT` *`string`* `[,` *`string`*`]` – Print string.

The following script demonstrates an application of these commands:

```
# Screen Display Demo
# Hide modem control strings from the user
#
.begin
  set @ONERROR = "CONTINUE"
  set @USERIF = "OFF"
  clear                 # Erase the screen
  cursor 12, 30
  put "Now connecting, please wait."
  set @SCRLREG = "OFF"
  connect
  set @SCRLREG = "ON"
  if @STATUS not = "0"
    set @USERIF = "ON"
    clear
    write "Can't connect or log in."
    return
  end
  terminal              # enter Terminal mode
  set @USERIF = "ON"    # don't forget this!
  return
# End of script.
```

Communicating with Other Programs

In some BLAST applications, the end user is not even aware that BLAST is operating in the system. BLAST provides a simple interface that lets other programs control BLAST, hiding the existence of BLAST completely from the user if necessary.

Passing Information to BLAST

The command line can contain up to ten "arguments," or parameters, that pass information to a BLAST script. Command line arguments follow the setup name on the command line (see "Command Line Switches" on page 10). For example, consider the following BLAST command line:

```
blast chicago -ssales 12:05 midwest
```

This command line will start BLAST with the chicago.su setup, execute the script called sales.scr using the `-s` switch, and store the arguments "12:05" and "midwest" in the reserved variables `@ARG0` and `@ARG1`, respectively.

A program can also pass information to a script by writing a text file that the script opens and interprets. Alternatively, because a script itself is just a text file, your controlling software can write a script that can be executed by BLAST "on the fly."

Controlling Other Programs from BLAST

While a script is executing, it can start other programs in your computer with the `LOCAL/SYSTEM` command. This command allows your script to execute a single command as you would type it on the command line. The following script demonstrates use of the `LOCAL/SYSTEM` command:

```
# Local System demo
# Copy a file
.begin
  set @syscmd = "cp modems.scr modems.txt"
  local
  system
  @syscmd
  esc
  return
# End of script.
```

File Transfers with BLAST Session Protocol

Chapter 6 describes the BLAST Session protocol, including some information about scripting file transfers. This section provides detailed information about writing these scripts.

The coding that performs a file transfer in a script closely follows the sequence of menu choices and prompts that BLAST uses when the same task is performed manually. Thus, it makes sense to practice a communications task interactively before attempting to write the script that will automate the task. Learn mode (page 176) provides another means of getting an idea about how a particular task can be coded in a script.

Getting and Sending Files

A simple `GET` and `SEND` could be coded like this (remember, you would *not* include the numbers in brackets):

```
[ 1] filetransfer
[ 2] get
[ 3] yourfile.rpt
[ 4] myfile.rpt
[ 5] ta
[ 6] send
[ 7] labdata.dat
[ 8]
[ 9]
[10] esc
```

In this script, yourfile.rpt (line 3) is the response to the Remote Filename prompt that BLAST issues when the `GET` command is given, and myfile.rpt (line 4) is the response to the Local Filename prompt. The transfer options `t` and `a` (line 5) specify "text" and "append" in this example—the same symbols you would use if you were performing the file transfer interactively. In the `SEND` example, two blank lines (lines 8 and 9) are entered to indicate that BLAST should use default values for these responses. Thus, the remote filename will be the same as the local filename, and no transfer options are specified (the file transfer will be binary). *Blank lines representing default filenames and file attributes (t, o, a) cannot contain comments.* Other than the preceding exceptions, you should not have blank lines in a script unless they *do* contain the comment character, #. The `ESC` statement represents pressing the *CANCEL* key, which is the action that you normally take to exit Filetransfer mode.

Performing Remote Commands

The BLAST session protocol allows you to perform remote system commands without special knowledge of the command syntax on the remote machine. Remote commands are coded in a script like this:

```
filetransfer
remote
  chdir
  /usr/customer
  esc
esc
```

The first ESC represents the escape keystroke that will move you from the Remote menu to the Filetransfer menu. The second ESC terminates the session in the usual manner.

Using Transfer Command Files

A powerful feature of the BLAST Session protocol is the ability to take its commands from a transfer command file (see "Transfer Command File" on page 115). To use a transfer command file in a script, the following syntax is used:

```
filetransfer
file
transfer.tcf
esc
```

where transfer.tcf is the command filename. The extension .tcf is often used to identify a transfer command file, but this convention is not required.

Sending Messages

BLAST protocol can send messages between systems during a BLAST session (see the description of the Message menu option on page 108). String-variables may be substituted for all elements except ESC.

```
filetransfer                  # issue the transfer command
message                       # sending a message
  Sending Sales Reports       # the message
esc                           # exit Filetransfer mode
```

Special Considerations

To take full advantage of the BLAST Session protocol, keep the following points in mind:

◊ BLAST attempts to queue as many remote commands as possible (like `GET`s) before issuing local commands (like `SEND`s). This behavior permits BLAST to transmit files in both directions simultaneously, but it also means that files may not be transmitted in the order specified in the script.

◊ Many filetransfer and file management commands can be combined into one `FILETRANSFER-ESC` block, as in the following example:

```
filetransfer         # begin Filetransfer mode
send                 # send files that
*.txt                # match the template
%
ta
remote               # begin remote file mgmt
  chdir
  /usr/customer
  print
  client.log
  esc                # leave remote file mgmt
file                 # use a command file
site3.tcf
esc                  # exit Filetransfer mode
```

Combining operations allows BLAST to work more efficiently, saving online charges or other long-distance telephone costs.

◊ Errors that occur during file transfer can be checked by testing the value of `@EFERROR` or by examining an `@EFLOG` file after exiting Filetransfer mode. If extended logging is enabled, additional reserved variables give information about the number of successful transfers and the number of failures. These reserved variables are described in Chapter 15. See also “Using Log Files for Error Checking” on page 205.

If the line is dropped during a file transfer, BLAST can either ignore the problem or abort Filetransfer mode immediately. The action BLAST takes is determined by the setting of the DCD Loss Response setup field, but the ability of BLAST to react to changes in DCD depends on the serial port device driver. If BLAST does not react to changes in DCD as expected, consult your system documentation for an alternate device driver that tracks DCD.

File Transfers with FTP Using 10.8x

The syntax for FTP file transfers is the same as for BLAST protocol except that there are no transfer options; therefore, there is no line for transfer options in FTP scripts. You may, however, add supported file transfer switches to receiving filenames (see “File Transfer Switches with FTP” on page 126). The basic file transfer syntax is:

```
filetransfer
send
local_filename
remote_filename
get
remote_filename
local_filename
esc
```

As with BLAST protocol, a blank line for the receiving filename indicates that the file will retain its original name. For example, in the following script

```
Filetransfer
get
Newinventory.txt

esc
```

the local filename will remain the same as the remote filename—Newinventory.txt.

File transfer scripts can be improved by adding error-checking features. For a discussion of error checking in file transfer scripts, see “Using Log Files for Error Checking” on page 205.

File Transfers with Kermit

Before writing scripts for Kermit, you may want to review the general information in Chapter 8, *Kermit Protocol*, on page 129. Learn mode (page 176) is also a good tool for obtaining a rough draft of a script you will need in a particular case.

Sending Files

Before issuing a SEND command, you must start simple Kermit or Kermit server on the remote machine.

Simple Kermit

After starting simple Kermit, you must issue a SEND command on the remote machine. The basic syntax for sending files using simple Kermit is as follows (the actual receive command depends on the specific implementation of simple Kermit)

```
Connect
TSEND "kermit", CR
TSEND "receive_command local_filename", CR
Filetransfer
SEND
local_filename
ESC
```

Kermit Server

Before issuing a SEND command, you must start Kermit server on the remote machine. For most UNIX machines, this command is "kermit -x." The basic syntax for sending files using Kermit server is as follows:

```
Connect
TSEND "kermit -x", CR
Filetransfer
SEND
local_filename
remote_filename
ESC
```

Receiving Files

Kermit has been implemented on many computer systems. BLAST's implementation of Kermit supports both "receiving" and "getting" files from remote computers. The RECEIVE command is used to transfer a file from simple Kermit, whereas a GET command is used for transferring a file from a Kermit server.

Simple Kermit

Before issuing a RECEIVE command, you must start simple Kermit on the remote machine and issue a send command. The basic syntax for receiving files using simple Kermit is as follows (the actual send command depends on the specific implementation of simple Kermit):

```
Connect
TSEND "kermit", CR
TSEND "send_command remote_filename", CR
Filetransfer
Receive
local_filename
ESC
```

Kermit Server

Before issuing a `GET` command, you must start Kermit server on the remote machine. For most UNIX machines, this command is “kermit -x.” The basic syntax for `GET`s using Kermit server is as follows:

```
Connect
TSEND "kermit -x", CR
Filetransfer
GET
local_filename
remote_filename
ESC
```

Transferring More Than One File

Unless you exit simple Kermit or Kermit server on the remote computer, you do not have to issue the command to start Kermit for every transfer block, only the first one. For example, you could run the following script:

```
Connect
TSEND "kermit -x”, CR
GET
SalesReport.txt
Store1Sales.txt
SEND
Store1Inventory.txt
Inventory.txt
ESC
TSEND "quit", CR
```

For simple Kermit, however, you do have to issue the simple Kermit send or get command each time you transfer a file, as in the following example:

```
Connect
TSEND "kermit", CR
TSEND "send SalesReport.txt", CR
Filetransfer
```

```
Receive
Store1Sales.txt
ESC
TSEND "receive Store1Inventory.txt", CR
Filetransfer
Send
Inventory.txt
ESC
TSEND "quit", CR
```

File transfer scripts can be improved by adding error-checking features. For a discussion of error checking in file transfer scripts, see “Using Log Files for Error Checking” on page 205.

File Transfers with Xmodem and Xmodem1K

Before writing scripts for Xmodem and Xmodem1K, you may want to review the general information in Chapter 9 on the use of these protocols. Learn mode (page 176) is also a good tool for obtaining a rough draft of the script you will need in a particular case.

Sending Files

Before issuing a `SEND` command, you must issue the Xmodem receive command on the remote computer for the remote system’s implementation of Xmodem. The basic syntax for sending a file using Xmodem is:

```
Connect
TSEND "receive_command remote_filename", CR
Filetransfer
SEND
local_filename
ESC
```

Receiving Files

The syntax for receiving files is:

```
Connect
TSEND "send_command remote_filename", CR
Filetransfer
GET
local_filename
ESC
```

Transferring More Than One File

A separate `FILETRANSFER-ESC` block is required for each file that is transferred. For example, to send two files and get one file, three `FILETRANSFER-ESC` blocks are needed, as in the following example:

```
# 3-File Xmodem Transfer
Connect
TSEND "rx Sales", CR
Filetransfer
SEND
S1Sales
ESC
TSEND "rx Order", CR
Filetransfer
SEND
S1Order
ESC
TSEND "sx Inventory", CR
Filetransfer
GET
S1Inventory
ESC
```

File transfer scripts can be improved by adding error-checking features. For a discussion of error checking in file transfer scripts, see “Using Log Files for Error Checking” on page 205.

File Transfers with Ymodem and Ymodem G

Before writing scripts for Ymodem and Ymodem G, you may want to review the general information in Chapter 9 on the use of these protocols. Learn mode (page 176) is also a good tool for obtaining a rough draft of the script you will need in a particular case. Because the filename is passed to the receiving computer, a filename is not needed when receiving a file.

Sending Files

Before issuing a `SEND` command, you must issue the Ymodem receive command on the remote computer for the remote system’s implementation of Ymodem. The basic syntax for sending a file using Ymodem is:

```
Connect
TSEND "receive_command", CR
Filetransfer
SEND
local_filename
ESC
```

Receiving Files

The syntax for receiving files is:

```
Connect
TSEND "send_command remote_filename", CR
Filetransfer
GET
ESC
```

Transferring More Than One File

A separate `FILETRANSFER-ESC` block is required for each file that is transferred. For example, to send two files and get one file, three `FILETRANSFER-ESC` blocks are needed, as in the following example:

```
# 3-File Ymodem Transfer
Connect
TSEND "rb", CR
Filetransfer
SEND
Sales
ESC
TSEND "rb", CR
Filetransfer
SEND
Order
ESC
TSEND "sb Inventory", CR
Filetransfer
GET
ESC
```

File transfer scripts can be improved by adding error-checking features. For a discussion of error checking in file transfer scripts, see "Using Log Files for Error Checking" on page 205.

Before writing scripts for Zmodem, you may want to review the general information in Chapter 9. Learn mode (page 176) is also a good tool for obtaining a rough draft of a script.

The Zmodem protocol is configured through the Zmodem setup subwindow. An important parameter for scripting purposes is Auto Receive. With Auto Receive set to `YES` in the setup file or the reserved variable `@ZMAUTODOWN` set to `YES` in a script, Zmodem will only receive files. Note that a setting for `@ZMAUTODOWN` in a script overrides the setting of Auto Receive in the setup file.

Because the filename is passed to the receiving computer, a filename is not needed when receiving a file.

Sending Files

Before issuing a `SEND` command, you must issue the Zmodem receive command on the remote computer for the remote system's implementation of Zmodem. In the basic syntax for sending a file using Zmodem below, the reserved variable for Auto Receive, `@ZMAUTODOWN`, is set to `NO` in case the Setup file has Auto Receive set to `YES` or `@ZMAUTODOWN` has been set to `YES` earlier in the session:

```
set @ZMAUTODOWN = "No"
Connect
TSEND "receive_command", CR
Filetransfer
SEND
local_filename
ESC
```

Receiving Files

The syntax for receiving files depends on the how you set `@ZMAUTODOWN`. If `@ZMAUTODOWN` is set to `NO`, you need a `GET` statement:

```
set @ZMAUTODOWN = "No"
Connect
TSEND "send_command remote_filename", CR
Filetransfer
GET
ESC
```

If @ZMAUTODOWN is set to YES, you do *not* need a GET statement

```
set @ZMAUTODOWN = "Yes"
Connect
TSEND "send_command remote_filename", CR
Filetransfer
ESC
```

Transferring More Than One File

As with Xmodem and Ymodem protocols, with Zmodem protocol each FILETRANSFER-ESC block can specify only one file, as in the following example:

```
set @ZMAUTODOWN = "No"
Connect
TSEND "rz", CR
Filetransfer
SEND
Sales.txt
ESC
TSEND "sz Inventory.txt", CR
Filetransfer
GET
ESC
```

File transfer scripts can be improved by adding error-checking features. For a discussion of error checking in file transfer scripts, see "Using Log Files for Error Checking" on page 205.

BLAST Operation as a Pseudohost With 10.8x

If a remote user logs onto your UNIX system and wants to perform a file transfer, the immediate question is: "What file transfer protocol will the remote operator use?" If he is running BLAST on his system, the best choice would be the BLAST Session protocol. In this case, the user starts BLAST with the command

```
blast -h
```

and then enters Filetransfer mode on his local system (see "Command Line Switches" on page 10). However, if the remote operator is not running BLAST but a session package that uses a public domain protocol, a "pseudohost" mode must be used. This mode is available for Xmodem, Ymodem, and Zmodem.

Pseudohost operation requires a special command line to start BLAST on the host system and to execute file transfers. The format of the command line for the remote user is:

blast [*setup*] -hs{x|k|y|g|z}*filename*
blast [*setup*] -hr{x|k|y|g|z}[*filename*]

where

setup – specifies setup file. Use this optional switch if you want to change the filetransfer parameters on the remote system.

-h – specifies host mode.

s *or* r – specifies either Send or Receive.

x|k|y|g|z – specifies either Xmodem, Xmodem1K, Ymodem, Ymodem G, or Zmodem protocol.

filename – specifies the file to be sent or received from the host UNIX. The filename must be specified for a Send. Wildcards can be used for Ymodem, Ymodem G, and Zmodem Sends. For Xmodem, the filename can be specified for a Receive; if it is not, the default filename will be /usr/tmp/XYZfile.

The following are examples of command line usage:

blast -hsxwill	sends the file named “will” using Xmodem.
blast -hrx	receives a file using Xmodem and saves it with the filename /usr/tmp/XYZfile.
blast ymg -hrg	starts BLAST on the remote system with the setup file named “ymg” loaded and receives a file using Ymodem G.

NOTE: *If the remote user accidently enters the wrong command line, there is no “graceful” exit as provided by the BLAST protocol. The terminal appears to hang until the protocol times out, which may take several minutes.*

Using Log Files for Error Checking

Checking for errors after a file transfer is an important part of a good script. Messages generated during a file transfer are written to the session log file, which you can open and read as you would any other file. For example, the following script automates a BLAST session and checks for errors:

```
  set @ftlog = "session.log"
  if exist @ftlog ldelete @ftlog
  set @LOGFILE = @ftlog
  filetransfer
  send
  orange
  fruit
  esc
  set @xferok = "NO"              # initialize user flag
  set @LOGFILE = ""               # close session log
  fopenr 1, @ftlog                # now open it for reading
.check
  fread 1, @logline
  if @STATUS = "0"                # successful read
    strinx @logline, "send complete"     # crucial!
    if @STATUS = "0" goto .check         # no match
    set @xferok = "YES"           # matched, set user flag
  end
  fclose 1
  if @xferok = "YES" display "Transfer successful"
  else display "Could not transfer the file"
  return                          # or whatever else
```

Another log file, the error-free log, is available for similar error checking. The error-free log, or “eflog,” contains just the status messages generated during a file transfer and is overwritten each time a FILETRANSFER-ESC block is executed, unlike the session log, which is always appended. Consequently, an eflog can be scanned more quickly than a session log because there are fewer lines to read and discard (see @EFLOG on page 270).

The following script fragment demonstrates how @EFLOG may be used to check for errors.

```
set @EFLOG = "xmodem.log"
filetransfer
get
portland.dat
esc
fopenr 1, @EFLOG   # check the log
fread 1, @input    # only 1 line to look at!
fclose 1
strinx @input, "ERROR"
if @STATUS = "0" display "No error occurred."
else display "Error!"
```

Text Transfers

The following section describes scripting for text transfers. See *Text Transfers* on page 145 for more information about text transfers.

Uploading Text

To upload a text file from within a script, write a BLAST script that includes:

◊ a `TSEND` command to start an editor to capture the data on the remote system and any commands needed for overwriting or appending the file.

◊ a `TUPLOAD` statement (this will honor the setup fields for flow control—XON/XOFF, Wait for Echo, Line Delay, Character Delay, Prompt Character—and linefeed handling). The `TUPLOAD` command sets `@STATUS` to `0` if successful; it returns some file I/O errors.

◊ a `TSEND` command to exit the editor on the remote system.

When uploading to a remote computer, remember that some of the data may be buffered. This means that the upload may complete well before all the characters have passed completely to the remote system. Any activity immediately following a `TUPLOAD` may have to deal with both the trailing characters of the uploaded file and the delay before other activity can be initiated. To avoid these problems, you can:

◊ `TTRAP` for the characters issued by the remote system upon exiting the text editor.

◊ Use a `WAIT IDLE` statement to be sure the buffers have a chance to clear.

The sample script below assumes that the remote computer is running UNIX using the text editor vi. The script `TTRAP`s for the filename in quotation marks used in vi's exit status line; the `WAIT` command gives the buffers on the local and remote computers time to clear.

```
connect
TSEND "vi cih4", CR     # Send cmd to start editor on remote
wait 3
TSEND "G", CR           # Moves cursor to end of file
```

```
TSEND "o", CR           # Starts new line for appending
TUPLOAD "cih4"
wait 3 idle
TSEND "\033", CR        # Send escape cmd to remote system
wait 1
TSEND ":x", CR          # Send cmd to exit editor on remote
TTRAP 30, "\042cih4\042"
set @hold = @status
wait 3 idle
if @hold = "0"
  display "Tupload not completed."
  return
end
else display "Tupload successful."
wait 10
```

For more specific error checking, you can check @STATUS for TUPLOAD:

```
connect
TSEND "vi cih4", CR
wait 3
TSEND "G", CR
TSEND "o", CR
TUPLOAD "cih4"
set @hold1 = @status
wait 3 idle
if @hold1 = "0" display "Tupload cmd execution complete."
else
  display "Tupload cmd failure; error ", @hold1
  TSEND "\033", CR
 TSEND ":q!", CR         # Quit editor without saving file
  return
end
TSEND "\033", CR
wait 1
TSEND ":x", CR
TTRAP 30, "\042cih4\042"
set @hold2 = @status
wait 3 idle
if @hold2 = "0"
  display "Tupload not completed."
  return
end
else display "Tupload completed."
wait 5
```

Downloading Text

To download a text file from within a script, write a BLAST script that includes a `TCAPTURE` statement. `TCAPTURE` will receive the specified file from the remote system and activate capture to receive it.

While `TTRAP` handles a small number of characters for processing by a BLAST script, `TCAPTURE` accepts large amounts of data and saves it to a disk file. The `APPEND` option writes the captured data to the end of an existing file or creates a new file. The `OVERWRITE` option deletes and recreates an existing file or creates a new file. If BLAST is unable to use the specified file, the statement will set `@STATUS` to an error code.

Once capture has been enabled, the program must execute one of the following statements before capture begins: `TERMINAL`, `TTRAP`, `TUPLOAD`, or `WAIT` (with `CARRIER` or `IDLE` option). To close the file and save any data that has been captured, use `TCAPTURE OFF`. The following example shows how a file can be displayed and captured from a remote computer:

```
connect
TSEND "cat payroll.dat", CR
TCAPTURE ON "payroll.cap"   # turn capture on
wait 5 idle        # wait for data to stop
TCAPTURE OFF       # end capture, close file
```

Chapter 14

Connecting and Disconnecting

Introduction

Connecting and disconnecting are crucial operations. Normally, BLAST initializes the modem and dials a remote system under the control of a specialized script called “modems.scr.” Logging into a remote system, such as a VMS or a UNIX-based computer, is likewise handled by a special script called “systems.scr.” These scripts are called by BLAST when the Connect command is issued from a menu or the `CONNECT` statement is executed in a script. Disconnecting is managed in a similar way by modems.scr and systems.scr. It’s important to understand the structure and operation of these two scripts—and how you can modify them.

BLASTscript Libraries

Modems.scr and systems.scr are called script “libraries” and provide the information that BLAST needs to control your modem and to log onto remote computers. These libraries are collections of scripts combined into large files and indexed for rapid access. BLAST automatically chooses the proper scripts from these libraries based on the values of the System Type and Modem Type setup fields. If you should choose to modify either modems.scr or systems.scr, *be sure to make a backup copy of the file first under another name.* As with any other script file, modems.scr and systems.scr should always be saved as text-only or ASCII files. *Do not save them as word-processor files.*

These script libraries are activated through menu commands or script commands, as follows:

Connect – Uses commands in modems.scr and systems.scr to dial out and log onto the remote system.

Upload – Uses commands in systems.scr to prepare the remote computer for the text upload.

Filetransfer – Uses commands in systems.scr to start BLAST on the remote computer.

Disconnect – Uses commands in modems.scr and systems.scr to log off the remote system and hang up the modem.

By automating these processes, BLAST allows you to exchange information between many different computer types without requiring technical proficiency in each system.

Modem Control

The modems.scr library handles a wide range of different modems, some of which may use proprietary commands to perform functions under computer control. BLAST uses the Modem Type setup field or the `@MODEM` reserved variable to select the proper script from this library and the Originate/Answer setup field or the `@ORGANS` reserved variable to tell the modem either to originate or to wait for calls.

Remote System Control

The systems.scr library controls the commands sent to the remote computer. By using this library, your system can start BLAST in host mode on the remote computer. BLAST also uses this library to control text uploading. BLAST uses the System Type setup field or the `@SYSTYPE` reserved variable to select the proper script from this library.

Creating New Libraries

You can create alternate system and modem control files that contain only the necessary commands for your particular hardware—this is more efficient than the standard libraries that include many modems and systems that you are not likely to need. BLAST will always look for individual files in the directory specified by the BLASTDIR environment variable before using the standard libraries. For example, if you specify `tblazer` in the Modem Type setup field or set `@MODEM` to `tblazer`, `CONNECT` will use a stand alone script named tblazer.scr, if it exists, to control modem handling instead of the tblazer entry in modems.scr.

The Connection Process in Detail

The modems.scr library can be used to automate the connect process. If the Modem Type setup field is empty or set to "hardwire," BLAST assumes that your system is hardwired to the remote computer and modems.scr is not opened.

When a Modem Type has been selected and the Originate/Answer setup field is set to `ANSWER`, control is passed to the `.ANSWER` section in modems.scr, which initializes the modem and waits for the call.

When the Originate/Answer field is set to `ORIGINATE` and the Connect command or `CONNECT` statement is used, control is passed to the `.DIAL` section. If a phone number is specified in the phone number field, `.DIAL` sends the phone number characters field to the modem as a dial command. If the Phone Number field is empty, `.DIAL` prompts the user to enter a number. After dialing, it waits for a message from the modem indicating a successful connection has been made.

If a System Type is specified, the corresponding `.LOGON` section in systems.scr is called for logging onto the remote system. If System Type is empty, BLAST assumes that you do not want system handling and the Connect process ends, returning you to the Online menu or the calling script with `@STATUS` set to `0`.

If an error is detected by modems.scr or systems.scr, the scripts return to BLAST with `@STATUS` set to reflect one of the errors listed below:

0	No error
1	Unable to initialize the modem (modems.scr)
2	No answer (modems.scr)

3 Can't log in: wrong userid, password (systems.scr)
4 No Carrier (modems.scr and systems.scr)
5 Busy (modems.scr)
6 No Dialtone (modems.scr)
7 Error (modems.scr)
8 OK unexpected (modems.scr)

Your script can check `@STATUS` to determine whether a connection is successful.

The Disconnection Process in Detail

There are four ways to disconnect from another system:

◊ You can select Terminal from the Online menu and manually type the appropriate commands to the modem and the remote computer.

◊ You can select Disconnect from the Online menu and allow BLAST to automate the process through the systems.scr and modems.scr libraries.

◊ You can write a BLAST script that uses the `DISCONNECT` statement, which operates similarly to the Disconnect command.

◊ You can physically hang up the modem by powering off. This is, of course, not recommended.

The Disconnect process attempts to log off the remote computer using the `.LOGOFF` section in systems.scr. Control is then transferred to the `.HANGUP` section in modems.scr to hang up the modem.

If an error is detected by modems.scr or systems.scr, the scripts return to BLAST with `@STATUS` set to reflect one of the errors listed below:

0 No error
1 Unable to initialize the modem (modems.scr)
3 Can't log out correctly (systems.scr)

Sample Modem Script

The following script illustrates the parts of a modem script. You can incorporate this script into modems.scr or keep it as a separate file, quick.scr. If you incorporate the script into modems.scr, you must index the script (see "The Index Utility" on page 216). If you incorporate and index the script, it will appear automatically as a new modem type in the Modem Type setup field. Otherwise, you must enter it manually into the Modem Type setup field.

```
:QUICK
#  A sample modem control script illustrating the
#  required sections .DIAL, .ANSWER, .HANGUP, and
#  .END.
#
.DIAL
  if NULL @PHONENO
    ask "enter phone number", @PHONENO
    if NULL @PHONENO or @STATUS = "-1" return 1
  end
  tsend "ATDT", @PHONENO, CR
  ttrap 45, "CONNECT", "NO CARRIER", "BUSY", "NO DIAL"
  if @STATUS = "1"
    ttrap "\015"
    return 0
  end
  let @STATUS = @STATUS + 2"  # set up return code
  return @STATUS
#
.ANSWER
  tsend "ATS0=1", CR
  ttrap "CONNECT"
  return 0
#
.HANGUP
  drop dtr
  wait 2
  raise dtr
  return 0
#
.END
:
#
# End of quick.scr
```

The required sections for a modem script are `.DIAL`, `.ANSWER`, `.HANGUP`, and `.END`. The appropriate section is activated when the Connect or Disconnect commands are given. The `.END` section terminates the script (or separates the script from the next one in modems.scr) and requires a final colon(:). With this sample, you should be able to write your own modem scripts or modify the scripts in modems.scr. Likewise, you can modify or enhance the system scripts in systems.scr.

Three files used by BLAST contain an index at the beginning of the file: blast.hlp, modems.scr, and systems.scr. Each index contains references to specific sections in the file. For instance, modems.scr contains a BLASTscript section to control the US Robotics Courier modem. The index at the beginning of modems.scr contains a reference to this section.

Indexing a file allows BLAST to jump to a particular section of a file quickly. Each section of the file should begin with a label in the form:

```
:LABEL
```

The index itself is in the form of lines of text, each beginning with the greater-than sign (>). The Index utility adds the numeric references that send control to the referenced section of the file.

If you modify any of these three files, the index must be recalculated so that BLAST can read the file properly. For example, if you add a new system type to systems.scr or add your own Online Help text to blast.hlp, you must run the index utility copied to your BLAST directory during installation to re-index the file. Indexing should only be performed on these three files. Before modifying or re-indexing any of these files, however, *be sure to make a backup copy of the file under another name and save the file you are modifying as text-only or ASCII.*

If you create a separate modem script, such as mymodem.scr and enter `mymodem` as the Modem Type in a setup, indexing is not required. If you modify any of the three standard files, however, you must re-index them. Follow this procedure to index a file:

1. Make a backup copy of the original file under another name.

2. Make the required changes to the original file.

3. Delete the old index lines from the file.

4. Save the file as text-only.

5. Rename the file.

6. Type the following command:

```
index oldfile newfile
```

where *oldfile* is the modified file and *newfile* is the name of the new indexed file. For example, if you modified systems.scr and saved it under the name sys.scr, you would type the following:

```
index sys.scr systems.scr
```

Remember also that BLAST will not operate properly if the final name of the file is not exactly as described above, that is, either systems.scr, modems.scr, or blast.hlp.

Chapter 15

BLASTscript Command Reference

Introduction

As you learned in Chapter 12, BLAST's script commands are English-like statements that automate communications functions. This chapter defines and illustrates the use of BLAST's script commands.

To use the script commands correctly, you must understand the data types supported by BLASTscript and the syntax rules defining a legal script statement.

Data Types

All data is stored as strings. The number of characters in a string is limited to the following:

10.7x	139 characters
10.8x	1,024 characters

Variables

Variables start with "@", followed by up to eight characters. For example:

```
@X
@Fred
@123
```

Names are not case-sensitive. Thus `@Fred`, `@fred`, and `@FRED` all refer to the same variable.

Numeric Constants

Numeric constants are sequences of digits enclosed in double quotation marks. They may not be preceded by a minus sign. For example:

```
"4"
"4789"
"56"
```

Numeric Strings

Numeric strings are sequences of digits enclosed in double quotation marks. Numeric strings may be preceded by a minus sign. For example:

```
"-4"
"4789"
"-56"
```

Numeric Values

Numeric values may be variables, numeric constants, or numeric strings as defined above.

String Constants

String constants are alpha-numeric sequences enclosed in double quotation marks. For example:

```
"THIS IS A STRING CONSTANT"
"12345"
".123ABC"
```

String constants may contain special control characters:

`\r`	carriage return
`\l`	linefeed
`\f`	formfeed
`\b`	backspace
`\t`	tab
`\\`	backslash character
`\xxx`	where *xxx* is the three-digit octal value of the character except for the octal value of null (`\000`), which is not permitted because null characters are treated as end-of-string characters. When encountered, nulls stop string processing.

Specifically, keep the backslash character in mind in writing scripts when your remote computer is a PC running DOS. If you quote a pathname, you will need to use double backslashes, as in the following example:

```
set @mydir = "\\DOS\\cih"
filetransfer
send
cih
@mydir

esc
```

If you want to include quotation marks in a `DISPLAY` or `WRITE` statement, a backslash must precede the quotation marks; otherwise, BLAST interprets the second quotation mark as the end of the string. For example, to display the following

```
Processing “Weekly Reports” -- please wait.
```

your script statement would be:

```
display "Processing \"Weekly Reports\" -- please wait."
```

Control characters may be coded in a string by preceding the character with “^”. For example, `^M` is equivalent to `\r` and `\015`:

```
set @msg = "3 carriage returns: ^M, \r, \015"
```

To code a single ^ in a string, two ^ characters are coded together.

String Values

String values may be string constants or variables as defined above.

Reserved Variables

Reserved variable values correspond to setup fields and physical or logical program conditions. See Chapter 16 for more information.

10.8x	**Binary Variables** Binary variables contain binary data. For example, the variable specified in a `HEX2BIN` command statement is a binary variable. Because these variables can contain nonprintable characters (nulls, for example), the contents of the variables may not display correctly on the screen.

Syntax Rules

The number of characters in a script statement is limited to:

10.7x	131 characters
10.8x	1,024 characters

Indentation makes code easier to read and has no effect on operation. Commands and variable names are not case-sensitive. Thus,

```
SET @FILENAME = "default.su"
```

is equivalent to

```
set @filename = "default.su"
```

If strings are numeric values, mathematical operations (+, -, *, /) can be performed in a `LET` statement. Parentheses are *not* allowed, however, and expressions are evaluated left to right without precedence.

Comment lines begin with “#”. Comments may also be placed on the same line as a BLASTscript statement by putting a # in the line; all characters from the # to the end of the line are treated as a comment.

Every line in a script must be executable or contain a comment. As a consequence, blank lines, which are rarely executable, cannot be used to separate script code visually.

BLASTscript is highly space-sensitive. When in doubt, separate all elements of a statement with spaces and enclose all constants, strings, or numerals in quotation marks. For example:

```
set @variable = "hello, world"
```

Commands That Set @STATUS

A number of script commands set the value of `@STATUS`, indicating whether the command was executed successfully. In general, `@STATUS` is set to `0` to indicate success. Some commands that return numeric results (e.g., `STRINX`, `TTRAP`) set `@STATUS` to `0` to indicate a null condition. The following commands set `@STATUS`:

ASCII	FOPENW	LOCAL SYSTEM	STRRINX
ASK	FREAD	LPRINT	STRLEN
CALL	FREADB	LRENAME	SYMTYPE
CONNECT	FREWIND	LTYPE	TCAPTURE
DISCONNECT	FWRITE	NEW	TSEND
DROP	FWRITEB	RAISE	TSENDBIN
FCLOSE	LCHDIR	REMOVE	TTRAP
FILETRANSFER	LDELETE	RETURN	TUPLOAD
FOPENA	LLIST	SELECT	WAIT CARRIER
FOPENR	LOAD	STRINX	WAIT IDLE

SET Statements that Set @STATUS

Additionally, `@STATUS` is set when you issue a `SET` command for a reserved variable that has a corresponding setup field; in this case, the change in the variable will occur immediately and will set `@STATUS` based on the success or failure of the change. For example, if you issue the following command:

```
set @PARITY = "ODD"
```

`@STATUS` will be set based on the success or failure of setting `@PARITY` to `ODD`.

10.8x Manipulation of Binary Data

BLAST Professional UNIX 10.8x permits manipulation of binary data using the reserved variables `@FILECNT`, `@SYMTYPE`, and `@TRAPCNT` (see Chapter 16) and the following BLASTscript commands— `BIN2HEX`, `CHECKSUM`, `FREADB`, `FWRITEB`, `HEX2BIN`, `SYMTYPE`, `TRAPNULLS_ON`, and `TSENDBIN`. For a full discussion of the function of these commands, see the description of specific commands below.

BLASTscript Commands

This section is organized alphabetically by command. The following conventions are used throughout:

[ ]	Indicates that enclosed phrases or characters are optional.
...	Indicates that the preceding statement or line may be repeated.
{ *xx* | *yy* }	Indicates that either the *xx* or *yy* phrase is required. Choose only one.
10.8x	Following the name of the command, indicates that the command is supported by 10.8x only.

ASCII

get ASCII value of a character

FORMAT: *ASCII string_value, numeric_value*

`ASCII` sets `@STATUS` to the ASCII value of the character at position *numeric_value* within *string_value*. The first position is 1. The ASCII value is the decimal value given to the ASCII character. For these values, see Appendix D.

EXAMPLE:

```
set @filename = "\\path\\filename"
ASCII @filename, 1           # get ASCII value for first
                             # character in @filename--
                             # ASCII 92 is a backslash (\)
if @STATUS = "92" display @filename, " is a full pathname"
```

ASK

prompt for a string from the user

FORMAT: *ASK [NOECHO] string_value, variable*

ASK prompts the user with *string_value* displayed at the top left of the screen. The input from the user will be placed in *variable*. Because of display limitations, the combined length of *string_value* and *variable* should not exceed 80 characters.

The NOECHO option causes BLAST to suppress user input. Use NOECHO when entering a password or other sensitive data. If the user replies to the ASK prompt by pressing ESC, @STATUS will be set to a nonzero value. If the input ends with ENTER, @STATUS will be set to 0 unless *variable* is a reserved variable that sets @STATUS in a SET statement. In this case, @STATUS is set based on the success or failure of the SET command. (see “SET Statements that Set @STATUS” on page 223).

EXAMPLE:

```
ASK "what month", @month
ASK NOECHO "Password?", @secret    # no display
```

BIN2HEX 10.8x

convert binary byte count to hexadecimal

FORMAT: *BIN2HEX numeric_value, variable1, variable2*

BIN2HEX converts the first number of bytes (*numeric_value*) in *variable2* into the hexadecimal equivalent of an ASCII string and stores the result in *variable1*.

NOTE: If *numeric_value* is larger than 512 bytes, the result of BIN2HEX will be too large for *variable1*. The size of *variable1* will always be twice as large as *numeric_value* because a binary character becomes a two-byte pair in hexadecimal.

EXAMPLE:

```
BIN2HEX 10, @buf, @arg1  # converts first 10 bytes of
                         # @arg1; stores result in @buf.
```

CALL

call another script

FORMAT: *CALL string_value*

CALL loads and executes another BLAST script, after which the called script returns to the calling script. *String_value* contains the filename of the called program. On return, @STATUS is set to the value of the exit code in the called program's RETURN statement or to 0 if no exit code value is given. Since all values are global, any values set in the calling script will be retained in the called script and vice versa. CALL searches for the script name in the following order:

1. Files without ".scr" extension in current working directory.
2. Files with ".scr" extension in current working directory.
3. Files without ".scr" extension in SETUPDIR directory.
4. Files with ".scr" extension in SETUPDIR directory.

EXAMPLE:

```
CALL "BACKUP.SCR"
if @STATUS = "0" display "Backup Successful"
```

CHECKSUM 10.8x

generate checksum of a string

FORMAT: *CHECKSUM numeric_value1, numeric_value2, var1, var2, [var3...]*
NOTE: *var=variable*

CHECKSUM generates a checksum or CRC from a string (*variable2* and any additional variables) and stores it as the hexadecimal equivalent of ASCII data in *variable1*. *Numeric_value1* specifies the type (checksum or CRC); *numeric_value2* specifies the compliment (0 or 1):

TYPE	COMPLIMENT
1 = 8-bit envoy LRC	0 = normal
2 = 16-bit CRC	1 = one's compliment
3 = 32-bit CRC	
4 = 8 checksum	
5 = 16 checksum	
6 = 32 checksum	
7 = Motorola pager 3-byte ASCII checksum	

EXAMPLE:

```
CHECKSUM 2, 0, @mylrc, @reply, @etx
#
# 16-bit CRC; 0 compliment; hexadecimal checksum of @reply
# and @etx is stored in @mylrc.
```

CLEAR

clear the scrolling region

FORMAT: *CLEAR*

`CLEAR` clears the scrolling region of the screen.

EXAMPLE:

```
CLEAR
```

CLEOL

clear to the end of the line

FORMAT: *CLEOL*

`CLEOL` clears from the current cursor position to the end of the current line in the scrolling region.

EXAMPLE:

```
CLEOL
```

CONNECT

connect to a remote

FORMAT: *CONNECT*

`CONNECT` directs BLAST to execute routines in the modems.scr and systems.scr libraries to dial the modem and log on if the Modem and System Type setup fields are specified. For more information about the operation of the `CONNECT` command, see Chapter 14.

EXAMPLE:

```
CONNECT
if @STATUS = "0" display "OK"
```

CURSOR

position the cursor within the scrolling region

FORMAT: *CURSOR numeric_value1, numeric_value2*

`CURSOR` positions the cursor to a given row (*numeric_value1*) and column (*numeric_value2*) in the 20 x 80 scrolling region. The row ranges from `0` to `19`, and the column ranges from `0` to `79`. If `@USERIF` is set to `0` or `OFF`, the full 24 x 80 screen will be addressed.

Use `PUT` statements following cursor position to write on the screen.

EXAMPLE:

```
CURSOR 4, 10                # move to row 4, column 10
put "1. Get sales figures"
CURSOR 6, 10
put "2. Send pricing"
ask "enter option (1 or 2)", @opt
```

DISCONNECT

disconnect from a remote

FORMAT: *DISCONNECT*

`DISCONNECT` directs BLAST to execute routines in systems.scr and modems.scr to log off and hang up the modem if the System and Modem Type setup fields are specified. See Chapter 14 for a full discussion.

EXAMPLE:

```
DISCONNECT
if @STATUS = "0" display "OK"
```

DISPLAY

display strings to display region

FORMAT: *DISPLAY string_value, ...*

`DISPLAY` displays messages in the scrolling region of the screen. If a log file has been specified, these messages will also be sent to the log file.

EXAMPLE:

```
DISPLAY "Dialing...", @PHONENO
```

DROP

drop DTR / RTS

FORMAT: *DROP { DTR | RTS }*

DROP terminates signals on the RS-232 interface. If the value is DTR, the Data-Terminal-Ready signal drops, hanging up most modems (cable and modem configuration permitting). If the value is RTS, the Request-to-Send signal drops, causing some devices to stop transmitting. The success of the DROP DTR and DROP RTS commands are dependent on the device driver being able to drop the signal on the serial port hardware.

EXAMPLE:

```
DROP DTR     # drop DTR signal
DROP RTS     # drop RTS signal
```

ECHO

enable/disable script display

FORMAT: *ECHO { ON | OFF }*

ECHO traces BLASTscript statements and displays them on the screen as they are executed. They are also echoed to the log file, if one is specified.

When executing CONNECT and DISCONNECT statements, the statements in the modems.scr and systems.scr libraries will also echo. If you do not wish to see all these statements, turn ECHO ON only as needed.

Because the statements displayed by ECHO are interspersed with the standard interactive dialog, ECHO is particularly useful in understanding what activity is triggered by what response within a BLAST script.

EXAMPLE:

```
ECHO ON      # set echo on
ECHO OFF     # set echo off
```

ERRSTR

store script error text

FORMAT: *ERRSTR numeric_value, string _variable*

ERRSTR puts the English language error message corresponding to *numeric_value* in *string_variable*. This statement is commonly used in association with the reserved variable @SCRIPTERR, which contains the number of the last BLASTscript error encountered.

For a list of error messages, see Appendix A. Note that not all error messages listed are possible errors in all versions of BLAST; some are operating system specific.

EXAMPLE:

```
fopenr 1, "nonexist.fil"
if @STATUS not = "0"
  ERRSTR @SCRIPTERR, @MESSAGE
  display "ERROR #", @SCRIPTERR, "-", @MESSAGE
end
```

FCLOSE

close an open file

FORMAT: *FCLOSE numeric_constant*

FCLOSE closes an open file. *Numeric_constant* is a number, called a handle, that other file statements use to refer to the file. The file handle can range from 1 to the number of file handles available through the operating system. If FCLOSE is successful, @STATUS is set to 0.

EXAMPLE:

```
fopenr 1, "input.fil"       # open file 1 for reading
FCLOSE 1                    # close file 1
```

FILETRANSFER FILE

perform commands from a BLAST TCF

FORMAT: *FILETRANSFER*
FILE
filename
ESC

In BLAST protocol, this multi-line statement performs commands read from a transfer command file (TCF). *Filename* is the name of a transfer command file, which may be specified with a string variable. See “Transfer Command File” on page 115 for a complete description of the transfer command file format.

EXAMPLE:

```
FILETRANSFER
FILE
command.fil
ESC
disconnect
quit
```

FILETRANSFER GET / SEND

get/send file

FORMAT:

FILETRANSFER	*FILETRANSFER*
GET	*SEND*
{protocol-dependent string(s) ...}	*{protocol-dependent string(s) ...}*
ESC	*ESC*

These statements transfer files to and from the remote computer. The exact syntax is protocol-dependent. For a full description of the syntax of the individual protocols, see “File Transfers with BLAST Session Protocol” on page 194 and the sections on scripting file transfers for the other supported protocols in Chapter 13.

EXAMPLE:

```
set @protocol = "BLAST"
set @new = "usr/blast/readme"
FILETRANSFER         # enter Filetransfer mode
GET                  # get a file with BLAST
getme.fil            # remote filename
@new                 # local filename stored in a variable
to                   # text conversion and overwrite
SEND                 # send a file with BLAST
*.DOC                # might be lots of these files...
%                    # resolve multiple names with %

SEND                 # send a file with no remote filename
samename.fil         # this will also be the remote name

t                    # send as text file
ESC                  # end BLAST protocol session
```

perform local commands using BLAST protocol

10.8x	perform local commands using FTP

FORMAT:

FILETRANSFER
LOCAL

{LIST |	*DELETE |*	*RENAME |*	*TYPE |*	*PRINT |*	*CHDIR |*	*SYSTEM}*
{SHORT|LONG}	*filename*	*oldname*	*filename*	*filename*	*pathname*	*command*
filename	*ESC*	*newname*	*ESC*	*ESC*	*ESC*	*ESC*
ESC	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*
ESC		*ESC*				

This multi-line statement performs Local menu commands within a `FILETRANSFER-ESC` block using BLAST protocol or, in BLAST Professional UNIX 10.8x, using FTP. Note that Local menu commands may also be performed with the `LLIST`, `LDELETE`, `LPRINT`, `LTYPE`, `LRENAME`, and `LCHDIR` statements.

`LOCAL` is followed by one or more commands. Most of the commands are followed by a filename, which may include wildcards or a string variable. Please note that lengthy local functions may force either the remote system or your system to time out, so keep local functions as short as possible or change the Inactivity T/O setup field to allow more time.

`LIST` – Display your local directory listing. The line after `LIST` must specify either `SHORT` or `LONG`. The second line after `LIST` can be left blank to display all files or it can be a filename, which may include wildcards (e.g., *.txt).

`DELETE` – Delete a file or files on your system. The line following `DELETE` is the filename, which may include wildcards.

`RENAME` – Rename a file on your system. The line after `RENAME` is the old filename; the second line after `RENAME` is the new filename.

`TYPE` – Type a file on your system's display. The line following `TYPE` is the filename.

`PRINT` – Print a file to the device specified in the `BPRINTER` environment variable. The line following `PRINT` is the filename (see `LPRINT` on page 247).

`CHDIR` – Change the working directory of your system. The line following `CHDIR` is the pathname of the new working directory.

SYSTEM – Perform a local system command. The line following SYSTEM is a system command. If this line is left blank, BLAST invokes the operating system interactively. When you are finished with the command interpreter, you must return to BLAST by typing exit and pressing ENTER. When BLAST is started with the -b switch (or with the -n switch if the display has not been re-enabled through a script), you cannot escape to a system prompt (see "Command Line Switches" on page 10).

EXAMPLE:

```
set @protocol = "BLAST"
FILETRANSFER          # start BLAST session protocol
get
daily.dat
new.dat
to
LOCAL                 # begin LOCAL commands
  PRINT
  new.dat
  RENAME
  new.dat
  old.dat
  ESC                 # end LOCAL commands
send
sendme.fil
toyou.fil
t
ESC                   # end BLAST protocol session
```

FILETRANSFER MESSAGE

send messages using BLAST Protocol

FORMAT: *FILETRANSFER*
MESSAGE
message
ESC

Using BLAST protocol, MESSAGE sends a text string that is displayed in the scrolling region of both computers' displays. The line after MESSAGE is a message—a line of text up to 67 characters or a variable containing a line of text up to 67 characters.

EXAMPLE:

```
FILETRANSFER                 # enter Filetransfer mode
MESSAGE                      # send a message
Sending Sales Reports        # specify the message
ESC
```

FILETRANSFER REMOTE

perform remote commands

FORMAT for BLAST protocol:

FILETRANSFER
REMOTE

{LIST |	*DELETE |*	*RENAME |*	*TYPE |*	*PRINT |*	*CHDIR |*	*MORE}*
{SHORT|LONG}	*filename*	*oldname*	*filename*	*filename*	*pathname*	*ESC*
filename	*ESC*	*newname*	*ESC*	*ESC*	*ESC*	*ESC*
ESC	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	
ESC		*ESC*				

This multi-line statement performs error-free file management on the remote computer during a BLAST protocol session. Multiple commands may follow the `REMOTE` command, and filenames (valid pathnames for the remote computer) or string variables may follow each command. Some older versions of BLAST do not support `REMOTE` commands.

During a BLAST session, the following commands are available:

`LIST` – Display the remote directory listing. The line after `LIST` must specify either `SHORT` or `LONG`. The second line after `LIST` can be left blank to display all files or it can be a filename, which may include wildcards (e.g., *.txt).

`DELETE` – Delete a file or files on the remote system. The line following `DELETE` is the filename, which may include wildcards.

`RENAME` – Rename a remote file. The line after `RENAME` is the old filename; the second line after `RENAME` is the new filename.

`TYPE` – Type a remote file on your system's display. The line following `TYPE` is the filename.

`PRINT` – Print a remote file to the remote printer. The line following `PRINT` is the filename.

`CHDIR` – Change the working directory on the remote computer. The line following `CHDIR` is the pathname of the new working directory

`MORE` – Continue displaying data from the remote computer after a page pause.

10.8x

FORMAT for FTP:

FILETRANSFER				
REMOTE				
{LIST	|	*DELETE*	|	*CWD}*
{SHORT | LONG}		*filename*		*pathname*
filename		*ESC*		*ESC*
ESC		*ESC*		*ESC*
ESC				

During an FTP session, the following commands are available:

`LIST` – Display the remote directory listing. The line after `LIST` must specify either `SHORT` or `LONG`. The second line after `LIST` can be left blank to display all files or it can be a filename, which may include wildcards.

`DELETE` – Delete a file or files on the remote system. The line following `DELETE` is the filename, which may include wildcards.

`CWD` – Change the working directory on the remote computer. The line following `CWD` is the pathname of the new working directory.

FORMAT for Kermit server protocol:

FILETRANSFER									
REMOTE									
{DIRECTORY |	*ERASE |*	*TYPE |*	*CWD |*	*SPACE |*	*WHO |*	*MESSAGE |*	*HOST |*	*KERMIT |*	*HELP}*
pathname	*filename*	*filename*	*pathname*	*pathname*	*user*	*message*	*command*	*message*	*ESC*
password	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*
ESC	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	*ESC*	
ESC									

During a Kermit server protocol session, the available commands depend upon both the version and the configuration of the remote Kermit server. A command may fail if the remote Kermit server does not support the command. You must start Kermit remote server on the remote system before entering Kermit Filetransfer mode. Kermit remote commands include:

`DIRECTORY` – Display a directory on the remote server. The line after `DIRECTORY` is the pathname (with or without wildcards) of the remote directory for which you want a listing; if you leave this line blank, the current working directory listing of the remote server will be displayed. The second line after DIRECTORY is the password that may be required to gain access to the directory listing. If no password is required, leave this line blank.

ERASE – Delete a file on the server. The line following ERASE is the filename (with or without wildcards) of the file to be erased. If you do not specify a full path for the file, the file (if it exists) will be removed from the current working directory of the remote server.

TYPE – Display a remote-server file on your screen. The line following TYPE is the filename of the file to be displayed. Kermit does not support a page pause, so you must use CTRL S to pause and CTRL Q to resume the flow of data.

CWD – Change the server's working directory. The line following CWD is the pathname of the new working directory.

SPACE – Display unused drive space of a directory on the remote server. The line following SPACE is the pathname (with or without wildcards) of the directory for which unused drive space is to be reported.

WHO – Display information on user(s) currently logged onto the server. The line following WHO is the user for whom you want information. If you leave this line blank, information on all users logged onto the server will be displayed.

MESSAGE – Send a one-line message to be displayed to the remote operator. The line following MESSAGE is the one-line message to be displayed to the remote operator.

HOST – Send an operating system command to the server. The line following HOST is the operating system command sent to the remote server. The command is executed immediately.

KERMIT – Send a Kermit language command to modify session parameters. The line following KERMIT is the message (Kermit language command) to be issued to the Kermit server, for example, SET FILE TYPE BINARY.

HELP – Display a short list of the available commands on the server.

EXAMPLE:

```
tsend "kermit -x", CR      # start kermit server on remote
FILETRANSFER               # enter Filetransfer mode
get
```

```
daily.dat
new.dat
REMOTE                      # start REMOTE commands
  CWD
  /usr/customer
  TYPE
  contactlist.txt
  ESC                       # end REMOTE commands
send
sendme.fil
toyou.fil
ESC                         # end Kermit protocol session
```

FLUSH

clear the input buffer

FORMAT: *FLUSH*

FLUSH clears the communications port input buffer. Only characters received after the FLUSH command has been executed will be available.

EXAMPLE:

```
FLUSH             # empty buffer
ttrap 10, "@"     # trap for "@"
```

FOPENA

open a file for appending

FORMAT: *FOPENA numeric_constant, string_value*

FOPENA opens a file for appending. If the file does not exist, it will be created. If it does exist, it will be opened and subsequent writes will append data to the end of the file. *String_value* is the filename of the file to be opened. *Numeric_constant* is a number, called a handle, that other file statements use to refer to the file. The file handle can range from 1 to the number of file handles available through the operating system. If FOPENA is successful, @STATUS is set to 0.

EXAMPLE:

```
FOPENA 1, "script.log"       # open file 1 for appending
fwrite 1, "got this far"     # adds string to the file
fclose 1                     # close file 1
```

FOPENR

open a file for reading

FORMAT: *FOPENR numeric_constant, string_value*

`FOPENR` opens a file for reading. The file must already exist. *String_value* is the filename of the file to be opened. *Numeric_constant* is a number, called a handle, that other file statements use to refer to the file. The file handle can range from `1` to the number of file handles available through the operating system. If `FOPENR` is successful, `@STATUS` is set to `0`.

EXAMPLE:

```
FOPENR 1, "command.fil"     # open file 1 for reading
fread 1, @input             # read the first line
fclose 1                    # close file 1
```

FOPENW

open a file for writing

FORMAT: *FOPENW numeric_constant, string_value*

`FOPENW` opens a file for writing. If the file does not exist, it will be created. If it does exist, all data in the file is overwritten. *String_value* is the filename of the file to be opened. *Numeric_constant* is a number, called a handle, that other file statements use to refer to the file. The file handle can range from `1` to the number of file handles available through the operating system. If `FOPENW` is successful, `@STATUS` is set to `0`.

EXAMPLE:

```
FOPENW 1, "cscript.log"      # open file 1 for writing
fwrite 1, "got this far"     # write string to file 1
fclose 1                     # close file 1
```

FREAD

read a line from a file

FORMAT: *FREAD numeric_constant, variable*

After an `FOPENR` command, `FREAD` reads a line of text into a variable. *Numeric_constant* is the file handle assigned the file in the `FOPENR` statement. If `FREAD` is successful, `@STATUS` is set to `0`. A nonzero value indicates an error reading the file or end of file.

EXAMPLE:

```
fopenr 1, "command.fil"        # open file 1 for reading
FREAD 1, @input                # read line into @input
if @STATUS not = "0"
  display "End of file reached"
end
fclose 1                       # close file
```

FREADB 10.8x

read a file as binary data

FORMAT: *FREADB numeric_value1, variable, numeric_value2*

FREADB reads up to a maximum number of bytes (*numeric_value2*) from the file specified by *numeric_value1* (the file handle assigned the file in the FOPENR statement) and stores the result in *variable*. The reserved variable @FILECNT stores the actual number of bytes read.

EXAMPLE:

```
FREADB 2, @line, 100  # reads up to 100 bytes from file
                      # handle 2 into @line.
```

FREE

release a variable from memory

FORMAT: *FREE variable*

FREE releases memory allocated to the specified variable. To recover all memory, you must FREE variables in the reverse order in which they were defined.

EXAMPLE:

```
FREE @input
```

FREWIND

rewind a file

FORMAT: *FREWIND numeric_constant*

FREWIND "rewinds" a file by resetting the file pointer to the beginning of the file. *Numeric_constant* is the file handle assigned the file in an FOPENR, FOPENW, or FOPENA statement. If FREWIND is successful, @STATUS is set to 0.

EXAMPLE:

```
fopenr 1, "commands.fil"    # open file 1 for reading
fread 1, @input             # read first line of file 1
FREWIND 1                   # rewind file 1
fread 1, @also              # read first line again
fclose 1                    # close file 1
```

FWRITE

write a line to a file

FORMAT: *FWRITE numeric_constant, string_value,...*

After an `FOPENW` command, `FWRITE` writes out a series of one or more strings to a file as a single line of text. *Numeric_constant* is the file handle assigned the file in an `FOPENW` or `FOPENA` statement. If `FWRITE` is successful, `@STATUS` is set to `0`.

EXAMPLE:

```
fopenw 1, "output.fil"
FWRITE 1, "the userid is: ", @USERID
fclose 1
```

FWRITEB 10.8x

write a file as binary data

FORMAT: *FWRITEB numeric_value1, variable, numeric_value2*

`FWRITEB` writes up to a maximum number of bytes (*numeric_value2*) from *variable* into the file specified by *numeric_value1*—the file handle assigned the file in an `FOPENA` or `FOPENW` statement. The reserved variable `@FILECNT` stores the actual number of bytes written.

EXAMPLE:

```
FOPENA 2, "sales.txt"
FWRITEB 2, @line, 100  # writes up to 100 bytes from @line
                       # into file 2.
```

GETENV

store the value of an environment variable

FORMAT: *GETENV string_value, variable*

`GETENV` writes the value of an environment variable (*string_value*) to *variable*.

EXAMPLE:

```
GETENV "BLASTDIR", @result
```

GOTO

branch to another point in program

FORMAT: *GOTO .LABEL*

`GOTO` branches unconditionally to another location in the program. `GOTO` will abort the program if `.label` cannot be found. The label is not case-sensitive and consists of eight characters or less, not counting the initial period.

EXAMPLE:

```
.PWD
  ask "enter the secret word", @pword
  if @pword = "rosebud" GOTO .CONT
  werror "invalid name"
  GOTO .PWD
.CONT
  display "Good morning, Mr. Phelps"
```

HEX2BIN 10.8x

convert hexadecimal to binary

FORMAT: *HEX2BIN numeric_value, variable1, variable2*

`HEX2BIN` converts the first number of bytes (*numeric_value*) in a hexadecimal string *(variable2*) into binary data and stores the result in *variable1*. *Variable1* will be one-half the size of *variable2* because each byte-pair will be reduced to one character.

EXAMPLE:

```
HEX2BIN 10, @buf, @arg1  # converts 1st 10 bytes of @arg1;
                         # stores result in @buf.
```

IF

perform single action if condition is true

FORMAT: *IF condition [{and / or}...] statement*

`IF` performs *statement* when *condition* is true. Evaluation is from left to right. Parentheses and arithmetic functions are not permitted in the condition.

The syntax of *condition* can be one of two forms. The first form is valid for string values only:

`string_value1 [NOT][>|>=|<|<=|=] string_value2`

The condition is true when `string_value1` is:

`>` greater than
`>=` greater than or equal to
`<` less than
`<=` less than or equal to
`=` equal to

`string_value2`.

The comparison is based on the ASCII values and the length of the strings. If the strings are not equal, the comparison is performed on the first different character in the strings.

The second form of the conditional clause is valid for numeric values only:

`numeric_value1 [NOT][GT|GE|LT|LE|EQ] numeric_value2`

The condition is true when `numeric_value1` is:

`GT` greater than
`GE` greater than or equal to
`LT` less than
`LE` less than or equal to
`EQ` equal to

`numeric_value2`.

Some special qualifiers provide an implied *condition*:

`[NOT]NULL string_value`
True [False] when `string_value` is of zero length.

`[NOT]numeric_constant`
True [False] when `numeric_constant` equals `@STATUS`.

`[NOT]REPS`
True [False] when the `REPS` counter is not zero (see page 186 for more information on using `REPS` and loops).

`[NOT]EXIST` *`string_value`*
True [False] when a file named the value of *`string_value`* exists.

10.8x	`[NOT] ISDIR` *`string_value`* True [False] when *`string_value`* is a directory.

`[NOT]OK`
True [False] when `@STATUS = "0"`.

EXAMPLE:

```
IF EXIST "file.one" LDELETE "file.one"
IF NOT NULL @VAR Display "@VAR is not empty"
IF @USERID = "FRED" GOTO .sendfiles
```

The following three statements are all equivalent:

```
IF OK GOTO .RUN
IF @STATUS = "0" GOTO .RUN
IF 0 GOTO .RUN
```

IF – ELSE

perform action for true or false conditions

FORMAT: *IF condition [{and / or}...] statement*
ELSE statement

`IF-ELSE` performs *statement* based upon *condition*. When the *condition* is true, the *statement* following the *condition* executes. When *condition* is false, the statement after `ELSE` executes. *Statement* must be on the same line as *condition*.

EXAMPLE:

```
connect
IF @STATUS = "0" write "Logged on successfully."
ELSE write "Logon failed!"
```

IF – END

perform multiple actions if condition is true

FORMAT: *IF condition [{and / or} condition...]*
statement
END

This multi-line clause performs several *statements* based upon *condition*. When the *condition* is true, subsequent statements up to the `END` are executed.

EXAMPLE:

```
IF @USERID NOT = "Annie"
  display "You can't run this script!"
  return 1
END
```

IF – END / ELSE – END

perform several actions for true or false conditions

FORMAT: *IF condition [{and / or} condition...]*
statement
END
ELSE
statement
END

This multi-line clause performs several *statements* based upon *condition*. When the *condition* is true, the *statements* up to the first `END` are executed. When the *condition* is false, the *statements* following `ELSE` and up to the `END` are executed.

When execution speed is important, use this statement instead of `GOTO`. Also, programs using this programming structure are generally easier to understand and maintain than programs using `GOTO`.

EXAMPLE:

```
ask "Ok to Log on?", @answer
IF @answer = "YES"
  display "Now Logging on"
  tsend @USERID, CR
END
ELSE
  display "Will not attempt to Log on"
  tsend "BYE", CR
END
```

LCHDIR

change working directory

FORMAT: *LCHDIR string_value*

`LCHDIR` changes the current working directory on the local computer to the directory specified in the *string_value*. If `LCHDIR` is successful, `@STATUS` is set to `0`.

EXAMPLE:

```
LCHDIR "work"              # change directory to work
if @STATUS = "0"           # if the return status is 0
  display "CHDIR ok"       # success!
end
```

LDELETE

delete a file on the local system

FORMAT: *LDELETE string_value*

`LDELETE` deletes from the local computer the file specified in *string_value*. If `LDELETE` is successful, `@STATUS` is set to `0`.

EXAMPLE:

```
LDELETE "sales.jun"
if @STATUS = "0"  display "sales.jun deleted"
```

LET

perform simple arithmetic

FORMAT: *LET variable = numeric value [{+ | – | * | /} numeric value]...*

`LET` does simple integer arithmetic. The expression is evaluated from left to right, with no grouping or precedence. The result is placed into a variable. The maximum and minimum integer values are 32,767 and negative 32,768.

When an integer becomes too large, the high order part of the number is discarded, resulting in unpredictable values. Fractional values after a division are always truncated.

EXAMPLE:

```
display "Polling statistics:"
LET @total = @numbad + @numgood
display "Total sites polled: ", @total
LET @next = @next + "1"
display "Next site is site number: ", @next
```

LLIST

display a listing of files on the system

FORMAT: *LLIST [LONG] string_value*

LLIST displays a directory listing on the local computer as specified by *string_value*. Wildcards may be used. If no path is given, items from the local current directory are listed. If LONG is specified, the listing will give all accompanying data rather than just the filenames and directory names.

10.7x	If the LLIST is successful, @STATUS is set to 0.
10.8x	@STATUS returns the number of items that match *string_value*.

EXAMPLE:

```
LLIST LONG "*"
Display @STATUS, " items are in the current directory."
```

LOAD

load a system setup

FORMAT: *LOAD string_value*

LOAD loads a setup from the directory specified by the SETUPDIR environment variable. *String_value* is the name of the setup. If the setup is in a subdirectory of the directory specified by SETUPDIR, the relative path must be included with the filename. The setup name should not include the .su extension. This statement operates like the Offline menu Select command and the SELECT statement. If the setup has been successfully loaded, @STATUS is set to 0.

EXAMPLE:

```
LOAD "Blaster"
if @STATUS = "0"
  display "Setup Blaster is the current setup"
end
else
  display "can't load the setup Blaster"
end
```

LOCAL SYSTEM

perform operating system command

FORMAT: *LOCAL*
SYSTEM
string_value
ESC

This multi-line statement performs local operating system commands. The line following SYSTEM is a system command. If this line is left blank, BLAST invokes the operating system interactively. When you are finished with the command interpreter, you must return to BLAST by typing exit and pressing ENTER. When BLAST is started with the -b switch (or with the -n switch if the display has not been re-enabled through a script), you cannot escape to a system prompt (see "Command Line Switches" on page 10).

EXAMPLE:

```
set @syscmd = "ls -l > catalog.txt"
LOCAL
SYSTEM
@syscmd
ESC
```

LOWER

convert variable to lowercase

FORMAT: *LOWER variable*

LOWER changes all uppercase characters in a variable to lowercase.

EXAMPLE:

```
ask "Enter your name:", @name
LOWER @name
```

LPRINT

print a file on the local printer

FORMAT: *LPRINT string_value*

LPRINT executes the command specified by the BPRINTER environment variable (see page 9). When BPRINTER specifies a target for printer output, LPRINT prints the file specified by *string_value* to that target. If the printer and file are found, @STATUS is set to 0 when the command specified by BPRINTER has been successfully executed.

EXAMPLE:

```
LPRINT "salesdata"
if @STATUS = "0" display "print worked ok"
```

LRENAME

rename a file on the local system

FORMAT: *LRENAME string_value1, string_value2*

LRENAME renames the local file specified in *string_value1* to the name specified in *string_value2* on the local computer. If the rename is successful, @STATUS is set to 0.

EXAMPLE:

```
LRENAME "f1.dat", "f2.dat"
if @STATUS = "0" display "Rename worked"
```

LTYPE

type a file on the local screen

FORMAT: *LTYPE string_value*

LTYPE types the local file specified in *string_value* on the screen. If the LTYPE is successful, @STATUS is set to 0.

EXAMPLE:

```
LTYPE "salesdata"              # display salesdata
if @STATUS = "0" display "LTYPE worked"
```

MENU

enable/disable menu display during script execution

FORMAT: *MENU {ON | OFF}*

MENU ON leaves the menu displayed for debugging purposes while a BLAST script is executing. Normally, menu display is suppressed during script execution.

EXAMPLE:

```
MENU ON     # set the menu display on
```

NEW

create a new BLAST setup

FORMAT: *NEW string_value*

`NEW` creates a new setup in the directory specified by the SETUPDIR environment variable (see page 10) based on the current values in memory. *String_value* is the name of the setup. If you want to put the setup in a subdirectory of the directory specified by SETUPDIR, the relative path must be included with the filename. The setup name should not include the .su extension.

The `NEW` statement operates like the Offline menu New command. If you specify a setup name that already exists, `NEW` will load that setup instead of creating a new one. If the setup has been successfully created, `@STATUS` is set to `0`; if an already existing setup has been loaded or there has been an error creating a new setup, `@STATUS` is set to `1`.

EXAMPLE:

```
NEW "CIS"                      # create setup named cis.su
if ok display "New setup created."
else display "Couldn’t create new setup."
```

PUT

output strings to the scrolling region

FORMAT: *PUT string_value,...*

`PUT` outputs one or more strings to the scrolling region. There is no implicit carriage return or new line after the output. This command is usually used in conjunction with the `CURSOR` statement.

EXAMPLE:

```
cursor 9, 30                  # put cursor in row 9,col 30
PUT "The winner is ", @win    # display string at
                              # cursor position
```

PWD

store the current path in a variable

FORMAT: *PWD variable*

`PWD` writes the present working directory location to a script variable.

EXAMPLE:

```
PWD @whereami
```

QUIT

quit BLAST and return to system with exit code

FORMAT: *QUIT numeric_constant*

`QUIT` aborts BLAST and returns to the operating system. *Numeric_constant* is an exit code that can be tested by the operating system.

EXAMPLE:

```
QUIT 123      # exit to operating system, exit status 123
```

RAISE

raise DTR/RTS

FORMAT: *RAISE {DTR | RTS}*

`RAISE` raises the Data-Terminal-Ready signal (`DTR`) or the Request-to-Send signal (`RTS`) on the RS-232 interface. These signals are normally used with modems. Some systems have `DTR` and `RTS` tied together so that raising either one affects both signals. The success of the `RAISE DTR` and `RAISE RTS` commands are dependent on the device driver being able to raise the signal on the serial port hardware.

EXAMPLE:

```
RAISE DTR                        # raise the DTR signal
RAISE RTS                        # raise the RTS signal
```

REMOVE

remove a system setup

FORMAT: *REMOVE string_value*

`REMOVE` deletes a setup from the directory specified by the SETUPDIR environment variable. *String_value* is the name of the setup. If the setup is in a subdirectory of the directory specified by SETUPDIR, the relative path must be included with the filename. The setup name should not include the .su extension. If the setup has been successfully removed, `@STATUS` is set to `0`.

EXAMPLE:

```
REMOVE "blaster"      # delete blaster.su
if @STATUS = "0" display "Setup blaster has been removed."
```

REPS

set repetition counter

FORMAT: *REPS numeric_value*

REPS creates loops in BLAST scripts. When REPS is used in an IF statement, it keeps track of the number of repetitions performed. The REPS numeric value is decremented and then tested for a value of zero. If *numeric_value* is a variable, the countdown occurs, but the variable retains its initial value.

EXAMPLE:

```
  REPS 3                      # loop three times
.loop
  display "hello"
  IF REPS GOTO .loop          # decrement; if REPS greater
  display "goodbye"           # than 0, branch to .loop;
```

RETURN

return to a calling program

FORMAT: *RETURN numeric_constant*

RETURN returns control to the menu system or the calling BLAST script. @STATUS of the calling script is set to *numeric_constant*, or 0 if no numeric constant is specified.

EXAMPLE:

```
RETURN 1    # return with @STATUS set to 1
```

SAVE

save a BLAST setup

FORMAT: *SAVE*

SAVE saves the current setup.

EXAMPLE:

```
SAVE                  # save current setup
```

SELECT

select a system setup

FORMAT: *SELECT string_value*

SELECT loads a setup from the directory specified by the SETUPDIR environment variable. *String_value* is the name of the setup. If the setup is in a subdirectory of the directory specified by SETUPDIR, the relative path must be included with the filename. The setup name should not include the .su extension. This statement operates like the Offline menu Select command. If the setup has been successfully loaded, @STATUS is set to 0.

EXAMPLE:

```
SELECT "Blaster"
If OK display "Setup successfully loaded."
Else display "Couldn’t load setup."
```

SET

set script variables to a string

FORMAT: *SET variable = string_value*

SET assigns a value to a variable. SET differs from the LET statement in that mathematical operations cannot be performed in a SET.

EXAMPLE:

```
SET @command = "blast -h"
SET @BAUDRATE = "9600"      # set baud rate in setup
SET @PARITY = "NONE"        # set parity in setup
```

SETTRAP

capture commport data to a script variable

FORMAT: *SETTRAP variable, numeric_constant1 [, numeric_constant2]*

SETTRAP prepares a TTRAP command to capture incoming data into a user-defined variable. Note that SETTRAP will not perform the capture itself—one or more TTRAPs must follow. Once a SETTRAP is issued, it remains in effect until another SETTRAP is issued; therefore, one SETTRAP can be used for multiple TTRAPs.

Variable specifies the destination for the TTRAP data. It may be either a new or previously used variable.

Numeric_constant1 defines the maximum number of characters to save into the variable. It must be greater than 0 and may be up to

10.7x	139 characters.
10.8x	1,024 characters.

Only the last incoming characters, specified by *numeric_constant1*, will be saved. When set to `0`, `SETTRAP` is disabled completely and the `TTRAP`(s) following will operate normally.

Numeric_constant2 contains the maximum amount of characters the `TTRAP`(s) will check for a match. If this value is reached, the `TTRAP`(s) will return to the calling script with `@STATUS` set to -5, and the `TTRAP` internal counter will be reset. Note that this is not on a per-`TTRAP` basis; the value is accumulated over one or more `TTRAP`s. This feature may be disabled by setting *numeric_constant2* to `0` or omitting it.

EXAMPLE:

```
# set TTRAP to capture data into @CAP--10 chars maximum;
# TTRAP exits if 85 chars are received before the TTRAP
# matches a string or times out
SETTRAP @CAP, 10, 85
TTRAP 6, "\015"       # trap next carriage return
SETTRAP @CAP, 20      # 20 chars placed in @CAP; no char
TTRAP 45, "Logout"    # count, so TTRAP will time out or
                      # match a string
```

STRCAT

combine strings

FORMAT: *STRCAT variable, string_value1[, string_value2...]*

`STRCAT` appends *string_value1* (and any additional string values) to *variable*.

EXAMPLE:

```
set @string1 = "abc"
set @string2 = "xyz"
STRCAT @string1, @string2    # append string2 to string1
display "alpha=", @string1   # display abcxyz
```

STRINX

find the first occurrence of one string in another

FORMAT: *STRINX string_value1, string_value2*

`STRINX` finds the first occurrence of *string_value2* in *string_value1*. `@STATUS` is set to the starting character position of *string_value2* in *string_value1*, or set to `0` if there is no match.

EXAMPLE:

```
set @string1 = "0123456"
STRINX @string1, "3"                # look for pattern "3"
display "The number 3 occurs at position ", @STATUS
```

STRLEN

determine the length of a string

FORMAT: *STRLEN variable*

`STRLEN` sets `@STATUS` to the length of *variable*.

EXAMPLE:

```
STRLEN @string
display "The length of @string is", @STATUS
```

STRRINX 10.8x

find the last occurrence of one string in another

FORMAT: *STRRINX string_value1, string_value2*

`STRRINX` finds the last occurrence of *string_value2* in *string_value1*. `@STATUS` is set to the starting character position of the last occurrence of *string_value2* in *string_value1*, or set to `0` if there is no match.

EXAMPLE:

```
set @string1 = "01234567890123456"
STRRINX @string1, "3"     # look for last occurrence of "3"
display "The number 3 occurs last at position ", @STATUS
```

STRTRIM

extract part of a string

FORMAT: *STRTRIM variable, numeric_value1, numeric_value2*

STRTRIM extracts a substring from *variable*. *Variable* is reset to the substring that begins at position *numeric_value1* and ends at position *numeric_value2*. If the original string will be required for further processing, a copy of it should be made before operating with STRTRIM, because STRTRIM changes the contents of *variable*.

NOTE: Make sure to include *numeric_value2*; if it is omitted, *variable* will contain nothing.

EXAMPLE:

```
set @name = "Anemometer"
STRTRIM @name, 4, 6
display "Hi,", @name
```

SYMTYPE 10.8x

reports the variable type

FORMAT: *SYMTYPE variable*

SYMTYPE determines the type (NONE, BINARY, STRING) of *variable* and reports the results in both @SYMBOLTYPE and @STATUS. @STATUS reports as follows:

```
0 = NONE (No variable of that name exists.)
1 = BINARY
2 = STRING
```

EXAMPLE:

```
SYMTYPE @arg1
```

TCAPTURE

enable text file capture

FORMAT: *TCAPTURE {ON [APPEND | OVERWRITE] | OFF} string_value*

TCAPTURE enables or disables text capturing while in Terminal mode. TCAPTURE ON enables Capture mode, and TCAPTURE OFF disables it. APPEND and OVERWRITE are used only with ON to indicate whether an existing file should be appended or overwritten. If neither is specified, APPEND is assumed.

@STATUS is set to 0 if *string_value* is a valid filename that can be written to; otherwise, @STATUS is set to an error code. TCAPTURE OFF does not affect @STATUS. No data is captured until one of the following is executed: TSEND, TTRAP, TUPLOAD, or WAIT with the CARRIER or IDLE option.

IMPORTANT: After issuing a TCAPTURE command, you should perform a WAIT IDLE or TTRAP to be sure that a stopping point has been reached in the data stream before exiting.

EXAMPLE:

```
TCAPTURE ON APPEND "test.cap"       # capture on; append
                                    # to file test.cap
if @STATUS not = "0"                # if not OK
  display "can't enable capture"    # write to screen
  return 1                          # return error code
end
tsend "cat bob.mail", CR            # send command to
                                    # the remote system
wait 10 idle                        # wait till no comm
                                    # port activity
TCAPTURE OFF                        # turn capture off
```

TERMINAL

become a terminal

FORMAT: *TERMINAL*

TERMINAL puts BLAST into Terminal mode, allowing the user to interact with the remote computer. Control cannot return to the script until the user types *ATTN ATTN*.
TERMINAL will not function if BLAST is started with the -b switch (batch mode) or -n switch (no display, unless the -n switch setting has been reset in the session—for example, in a script with the following command: SET @SCRLREG = "ON").

EXAMPLE:

```
display "Script paused..."
TERMINAL
display "Script continuing..."
```

TRAPNULLS OFF 10.8x

disable null traps

FORMAT: *TRAPNULLS_OFF*

`TRAPNULLS_OFF` disables the trapping of nulls; disabling null traps is the default mode.

EXAMPLE:

```
TRAPNULLS_OFF
```

TRAPNULLS ON 10.8x

enable null traps

FORMAT: *TRAPNULLS_ON*

`TRAPNULLS_ON` enables trapping of nulls (0x00's) in order to trap binary CRC's or checksums.

EXAMPLE:

```
TRAPNULLS_ON
```

TSEND

send strings to the remote computer

FORMAT: *TSEND {BREAK | CR | LF | string_value},...*

`TSEND` sends breaks, carriage returns, line feeds, or strings to the remote computer. Any combination of strings, line terminating characters, and/or breaks can be sent.

NOTE: Some operating systems (including DOS) expect a `CR/LF` instead of a `LF` at the end of a line. Take this into consideration and use `CR/LF` instead of `LF` for these systems. You might define an end-of-line variable at the beginning of a BLAST script to make these programs easily transportable to other systems.

EXAMPLE:

```
set @endline = "CR"
TSEND BREAK                          # send break signal
TSEND "ATDT", @PHONENO, @endline     # dial the modem
```

TSENDBIN

convert hexadecimal to binary while transmitting to remote

FORMAT: *TSENDBIN variable1 [,variable2...]*

`TSENDBIN` converts the hexadecimal equivalents of an ASCII string (*variable*) to binary code as they are sent to the remote system. Variables must contain hexadecimal strings.

EXAMPLE:

```
TSENDBIN @arg1, @arg2
```

TTRAP

trap for output from the remote computer

FORMAT: *TTRAP [MM:SS | SS,] string_value1 [,...string_value8]*

`TTRAP` pauses the BLAST script in Terminal mode, testing data flow to the communications port. When `TTRAP` sees one of the string values, it continues to the next statement. If *mm:ss* (minutes:seconds) is given and none of the string values is received in that length of time, `TTRAP` times out. `TTRAP` sets `@STATUS` to the number of the string that was found, or sets `@STATUS` to `0` if `TTRAP` timed out.

EXAMPLE:

```
set @x = "NO CARRIER"
TTRAP 30, "CONNECT", @x
if @STATUS = "0" write "Timeout on trap"
if @STATUS = "1" write "Connected!"
if @STATUS = "2" write "No carrier!"
```

TUPLOAD

upload a text file to the remote system

FORMAT: *TUPLOAD string_value*

`TUPLOAD` opens the file specified by *string_value* and sends the text to the remote computer. The transmission is paced by any flow control options specified in the setup. `TUPLOAD` sets `@STATUS` to `0` on completion of the text upload. If the upload is unsuccessful, `@STATUS` is set to the applicable BLAST error code. For example, if the file could not be found, `@STATUS` is set to `51` (error opening data file).

Some device drivers buffer the flow of data extensively. This means the TUPLOAD statement may complete well before all the characters clear the local and remote computer buffers.

NOTE: After a TUPLOAD command has been issued, it is a good idea to TTRAP for characters signaling the end of the upload or do a WAIT mm:ss IDLE. Exiting BLAST before the buffers are emptied may cause BLAST to terminate abnormally. See "Uploading Text" on page 207.

EXAMPLE:

```
connect
tsend "vi Sal", CR      # Send cmd to start editor on remote
wait 3
tsend "G", CR            # Moves cursor to end of file
tsend "o", CR            # Starts new line for appending
TUPLOAD "Sal"
wait 3 idle
tsend "\033"              # Send escape cmd to remote system
wait 1
tsend ":x", CR            # Send cmd to exit editor on remote
ttrap 30, "\042Sal\042" # trap filename in exit status line
set @hold = @status
wait 3 idle
if @hold = "0"
  display "Tupload not completed; error ", @hold
  return
end
else display "Tupload successful"
wait 10
```

UPPER

convert a variable to uppercase

FORMAT: *UPPER variable*

UPPER changes all lowercase characters in *variable* to uppercase.

EXAMPLE:

```
UPPER @salesdata
```

WAIT

wait for time to pass

FORMAT: *WAIT {MM:SS | string_value}*

`WAIT` pauses the BLAST script for *mm* minutes and *ss* seconds. *String_value* must be in the format *mm:ss*. The maximum value is 60 minutes (`60:00`).

EXAMPLE:

```
WAIT 2:02       # wait two minutes, two seconds
WAIT 2          # wait two seconds
WAIT 60:00      # wait one hour
```

WAIT CARRIER

wait for a phone call

FORMAT: *WAIT {MM:SS | string_value} CARRIER*

`WAIT CARRIER` pauses the BLAST script *mm* minutes and *ss* seconds, or until the modem raises carrier detect. If the modem raises carrier detect, `@STATUS` is set to `0`. If the statement times out, `@STATUS` is set to a nonzero value. The maximum value is 60 minutes (`60:00`). Carrier detection may not be available on some communications ports if the device driver does not provide the signal. Make sure that the modem and cable are configured to indicate when the carrier signal is present.

EXAMPLE:

```
WAIT 2:02 CARRIER     # wait two minutes and
                      # two seconds for a call
WAIT 12:00 CARRIER    # wait 12 minutes for a call
WAIT 12 CARRIER       # wait 12 seconds for a call
```

WAIT IDLE

wait for communications port activity to finish

FORMAT: *WAIT {MM:SS | string_value} IDLE*

`WAIT IDLE` pauses the script until no characters are received on the communications port for *mm* minutes and *ss* seconds. The maximum value is 60 minutes (`60:00`).

EXAMPLE:

```
WAIT 2:02 IDLE          # wait for two minutes and
                        # two seconds of idle
WAIT 1:00 IDLE          # wait for one minute of idle
WAIT 1 IDLE             # wait for one second of idle
```

WAIT UNTIL

wait for a specified time of day

FORMAT: *WAIT UNTIL {HH:MM | string_value}*

`WAIT UNTIL` pauses the script until the time is *hh* hours (24-hour clock) and *mm* minutes.

EXAMPLE:

```
WAIT UNTIL 2:02         # wait till 2:02 am
WAIT UNTIL 1:00         # wait till 1:00 am
WAIT UNTIL 13:30        # wait until 1:30 pm
```

WERROR

write an error message to the second menu line

FORMAT: *WERROR string_constant*

`WERROR` writes an error message to the operator and the log file. If `@ONERROR` is set to the default setting, `STOP`, `WERROR` pauses for a key to be pressed before continuing. Do not use this statement when writing a BLAST script that will be unattended unless `@ONERROR` is set to `CONTINUE`.

EXAMPLE:

```
WERROR "no response"         # display error message
return 1                     # return with @STATUS set to 1.
```

WRITE

write a message to the second menu line

FORMAT: *WRITE string_constant*

`WRITE` displays a message to the operator and the log file (without pausing as in `WERROR`).

EXAMPLE:

```
WRITE "dialing CHICAGO"
```

Chapter 16

BLASTscript Reserved Variables

BLASTscript reserved variables are an important part of any program that tests the condition of the communication session or the results of other statements.

There are two types of BLASTscript reserved variables: read-only and read/write. BLAST scripts can test a physical signal or logical condition using read-only variables. With read/write variables, scripts may not only test but also change a condition by using the `SET` command.

Reserved variables that reflect multiple-choice setup fields may be `SET` by using the value offered by the setup field. For example,

```
SET @DCDLOSS = "ABORT"
```

will change the value of the DCD Loss Response setup parameter in the BLAST protocol to `ABORT`.

In the following descriptions, if the reserved variable is associated with a setup field, the setup field will be indicated by italic print as the last line of the variable description. The characteristics of such fields are described in Chapter 5. The default value of the reserved variable is indicated by bold print and brackets.

@7BITCHN

read/write
YES **[NO]**

For BLAST protocol transfers, defines the data-path width.

BLAST Protocol subwindow: 7-Bit Channel

@ACKFREQ

read/write
1 – window size **[4]**

For BLAST protocol transfers, specifies the frequency at which an acknowledgement from the receiving system is requested. The frequency is measured in number of packets sent. See also `@WDWSIZ` (page 295).

BLAST Protocol subwindow: Ack Request Frequency

10.7x

@ANSIAUTOWRAP

read/write
YES **[NO]**

For PC ANSI emulation, selects automatic wrapping of lines longer than 80 characters.

ANSI Emulation subwindow: Auto Wrap

@ANSILEVEL

read/write
2.x **[3.x]**

For PC ANSI emulation, selects the correct level of ANSI for your system. Some applications require ANSI Level `2.x`.

ANSI Emulation subwindow: ANSI Level

@APROTO

read/write
YES **[NO]**

For BLAST protocol transfers, specifies whether the BLAST "A" Protocol will be used. Set this field to YES to communicate with older versions of BLAST.

BLAST Protocol subwindow: Use "A" Protocol

@ARG*n*

read/write
user-defined

Stores variables passed from the operating system command line. This variable is a read-only variable where *n* specifies the argument, from `0` to `9` (`@ARG0`, `@ARG1`, etc.). The command line must include a setup name before the first command line parameter is given (see "Command Line Switches" on page 10).

@ATTKEY

read/write
any Control Key **[^K]**

Defines the attention key (*ATTN*).

10.8x	Setting this variable to null (`@ATTKEY = ""`), turns off the *ATTN* key, for example during the running of a script. The *ATTN* key remains off until `@ATTKEY` is reset or until the script ends (or until the masterscript ends if one or more scripts are called), at which time BLAST resets `@ATTKEY` to its previous setting.

Setup field: Attention Key

@AUTOLFIN

read/write
YES **[NO]**

When set to `YES`, forces BLAST—while in Terminal mode—to insert a linefeed character after every carriage return character displayed.

Setup field: AutoLF In

@AUTOLFOUT

read/write
YES **[NO]**

When set to `YES`, forces BLAST—while in Terminal mode—to insert a linefeed character after every carriage return that leaves the communications port.

Setup field: AutoLF Out

@BAUDRATE

read/write
300 600 1200 2400 4800
[9600] 19.2 38.4 57.6 115K

Specifies the serial port device driver speed. The default value of this variable is set during the BLAST installation process. Some systems may not support higher baud rates.

Setup field: Baud Rate

@BLASTDIR

read-only

Specifies the directory path for the BLAST support files as defined in the BLASTDIR environment variable (see “Environment Variables” on page 7).

@CHARDLY

read/write
[0] – 999

Specifies the time delay (in hundredths of a second) between each character sent to the remote computer when uploading text or executing `TSEND` commands.

Setup field: Char Delay

@CLASS

read-only

Stores the BLAST class number of the local system.

@COMMPORT

read/write
any valid device

Stores the specification for the communications port, host system for TCP/IP connections, or hunt file that BLAST will use for the current session. Valid options are:

Device name – Any valid asynchronous port (e.g., /dev/tty1A).

Host name or address – The name or network address of the TCP/IP host system to which you want to connect (for example, “blaster.blast.com”). To establish a raw socket, set `@COMMPORT` to the host name and any available port number except 23. Port number 23 is reserved for telnet. To use telnet, simply give the host name, and BLAST will default to port number 23. To use telnet with a port other than port 23, give the host name, the port number, and “telnet,” as in the example below:

```
set @COMMPORT = "blaster.blast.com 12 telnet"
```

See “Accessing TCP/IP Ports” on page 16.

Hunt filename – The name (including path) of a hunt file that lists available devices preceded by the “<” character. Refer to “Automatic Serial Port Searching” on page 25 for details about hunt files.

Setup field: Connection

@COMP_LVL

read/write
0 – 6 **[4]**

For BLAST protocol transfers, specifies the maximum sending and receiving compression levels to be used. Level `0` specifies no compression; level `6` specifies the highest level of compression. Setting this variable is effectively equal to setting both the `@RCOMP_LEV` and `@SCOMP_LEV` reserved variables.

@CONNTIMO

read/write
0 – 999 **[60]**

Specifies the number of seconds BLAST will wait for a network connection. This field has no effect on serial connections.

Setup field: Connection T/O

@CONTIMO

read/write
0 – 999 **[120]**

Used with older versions of BLAST. For BLAST protocol transfers, specifies the time interval (in seconds) that BLAST will wait for a packet of data from the remote computer before timing out.

IMPORTANT: This reserved variable has been replaced by the reserved variable `@INACTIMO` and should not be used. Do not confuse it with the `@CONNTIMO` reserved variable described directly above.

@CTS

read-only

Stores the Clear-to-Send (CTS) device status. If `@CTS` is set to `1`, the device, usually a modem, is ready to receive characters. `@CTS` is set to `0` if the device is not ready to receive characters. The value of this variable is valid only if the serial port device driver returns the correct code.

@D/S_BITS

read/write
7/1 7/2 **[8/1]** 8/2

Sets data and stop bits for the communications port.

Setup field: Data/Stop Bits

@DATE

read-only

Contains the current date. By default the format is *mm/dd/yy*. This format may be changed using the reserved variable `@DATEFORMAT` or one of the following switches:

10.7x	-2
10.8x	-dd or -y

See "Command Line Switches" on page 10.

This is a read-only variable; an error message will be displayed if a script attempts to write to it.

@DATEFORMAT

read/write
template

Sets the format of the `@DATE` variable. Setting the `@DATEFORMAT` reserved variable overrides the format in which BLAST was started. The format of the output of the `@DATE` reserved variable will be determined by the `@DATEFORMAT` template set by the user. The value of the replacement sequences are as follows:

`%A` full weekday name (`Monday`)
`%a` abbreviated weekday name (`Mon`)
`%B` full month name (`January`)
`%b` abbreviated month name (`Jan`)
`%c` standard date/time representation (`%a %b %d %H:%M:%S %Y`)
`%d` day-of-month (`01-31`)
`%H` hour (24 hour clock) (`00-23`)
`%I` hour (12 hour clock) (`01-12`)
`%j` day-of-year (`001-366`)
`%M` minute (`00-59`)
`%m` month (`01-12`)
`%p` local equivalent of AM or PM
`%S` second (`00-59`)
`%U` week-of-year, first day Sunday (`00-53`)
`%W` week-of-year, first day Monday (`00-53`)

%w weekday (0-6, Sunday is 0)
%X standard time representation (%H:%M:%S)
%x standard date representation (%a %b %d %Y)
%Y year with century
%y year without century (00-99)
%Z time zone name
%% percent sign

For example, to set @DATEFORMAT to generate a date in the format of 19-March-1998, your script would read

```
set @DATEFORMAT = "%d-%B-%Y"
```

@DCD read-only

Stores the Carrier-Detect status from the modem. If @DCD is set to 1, the carrier is detected by the modem. If @DCD is set to 0, the modem does not sense a carrier from another modem. The modem must be set appropriately for this variable to reflect the state of the data carrier; and the modem cable, if present, must have the appropriate conductor. The value of this variable is valid only if the serial port device driver returns the correct code.

@DCDLOSS read/write

ABORT **[IGNORE]**

For BLAST protocol transfers, specifies whether BLAST will ABORT after or IGNORE DCD loss. This feature requires appropriate modem initialization and recognition of the signal by the serial port device driver (see discussion of @DCD above).

BLAST Protocol subwindow: DCD Loss Response

@EFERROR read/write

For BLAST protocol, returns the error code of the last error in a file transfer (see Appendix A). If no error occurs during the BLAST session, @EFERROR will remain set at 0. @EFERROR should be reset to 0 for continued testing during a session. Because BLAST queues filetransfer requests and then continues execution until ESC is encountered, testing @EFERROR within a FILETRANSFER-ESC block may not produce expected results.

Following completion of a BLAST protocol file transfer, @EFERROR will be set to a transfer file management error (error 31–49; see

"Transfer File Management" on page 340) or one of the following values reflecting the way in which Filetransfer mode was exited:

`0`	No errors
`-1`	Initialization error
`-2`	Local operator ended activity with *ATTN*
`-3`	Remote disconnect
`-4`	Never got starting message (Logon Timeout)
`-5`	Lost communications with remote system (Inactivity Timeout)
`-6`	Private network error; private network version of BLAST required
`-7`	DCD loss during Filetransfer logon
`-8`	DCD loss during Filetransfer session

Example:

```
connect
set @protocol = "BLAST"  # BLAST protocol only!!
set @EFERROR = "0"
filetransfer
send
test1.fil
recv1.fil
to
esc
if @EFERROR not = "0"
  display "Error number = ", @EFERROR, "occurred"
  display "See Chapter 16 and Appendix A for details."
  set @EFERROR = "0"
end
disconnect
return 0
```

@EFLOG

read/write
filename

Specifies a separate error-free log file that will log all filetransfer session errors or completions, or both, depending on the setting of `@EFLOGGING`. The default of `@EFLOGGING` is `BOTH`. Setting `@EFLOG` to a valid filename starts filetransfer session logging in `BOTH` mode. Setting `@EFLOG = ""` (null) turns off filetransfer session logging. The information written to the file appears exactly as it does on the user's screen, allowing easier parsing of a filetransfer session.

@EFLOGGING

read/write
[BOTH] ERRORS
COMPLETIONS

Specifies whether the log file named in `@EFLOG` will log filetransfer `ERRORS`, `COMPLETIONS`, or `BOTH`. Refer to `@EFLOG` above for further information.

@ELAPTIME

read-only

Contains the current elapsed online time for a BLAST communications session. The value is in *hh:mm:ss* format. This variable can be reset within a BLAST script by any `SET` statement, for example:

```
set @ELAPTIME = "it doesn't matter"
```

The current value is not checked and is simply reset to 00:00:00.

@EMULATE

read/write

10.7x	**[VT320]** any valid terminal emulator Specifies the terminal type to emulate in Terminal mode. Acceptable values are `VT320`, `VT220`, `VT100`, `VT52`, `PC ANSI`, `TV920`, `D80`, `ADM3A`, `WYSE60`, `WYSE50`, `TTY`, and `PASSTHRU`.
10.8x	TTY and **[PASSTHRU]** Specifies the terminal type to emulate in Terminal mode. Acceptable values are `TTY` and `PASSTHRU`.

Setup field: Emulation

@ENABLEFS

read/write
YES **[NO]**

For BLAST protocol transfers, enables the /FWD and /STR file transfer switches, which automatically delete files.

BLAST Protocol subwindow: Enable /FWD and /STR

@ENABLERCMD

read/write
[YES] NO

For BLAST protocol transfers, enables the /OVW (overwrite) file transfer switch and allows system commands to be sent from the remote system.

BLAST Protocol subwindow: Enable /OVW and Remote Cmds

10.8x	**@FILECNT** read-only Returns the number of bytes either written or read during `FREAD`, `FWRITE`, `FREADB`, and `FWRITEB`.

@FILTER

read/write
ON **[OFF]**

For BLAST protocol transfers, specifies whether the protocol filter is turned on. When `@FILTER` is set to `ON`, BLAST strips VT sequences sent from a mainframe protocol converter, preventing BLAST protocol from labeling these as bad blocks.

BLAST Protocol subwindow: Filtering

@FULLSCR

read/write
[YES] NO

Specifies whether the top four lines of the BLAST menu region will be suppressed while in Terminal mode. Set to `YES` to suppress the menu and `NO` to enable it.

Setup field: Full Screen

@INACTIMO

read/write
0 – 999 **[120]**

For BLAST protocol transfers, specifies the time interval (in seconds) that BLAST will wait for a packet of data from the remote computer before timing out.

NOTE: This variable replaces the `@CONTIMO` variable of previous versions.

BLAST Protocol subwindow: Inactivity T/O

@KBCHECK

read/write
1 – 3 **[2]**

For Kermit transfers, specifies the level of error-detection.

Kermit Protocol subwindow: Block-Check-Type

@KDELAYOS

read/write
1 – 99 **[5]**

For Kermit transfers, specifies the number of seconds of delay between the recognition of a Send command and the actual beginning of the transmission.

Kermit Protocol subwindow: Delay

@KEYBOARD

read/write
[ON] OFF

Controls the ability to enter data from the keyboard. If `ON`, the keyboard is unlocked and may be used. If `OFF`, BLAST ignores any keyboard characters, for example, during the running of a script to prevent extra characters from being sent in Terminal mode. After the script has run (or the masterscript ends if one or more scripts are called), BLAST resets the value of `@KEYBOARD` to the default, `ON`. When started in video-suppress mode (`-n` command line switch), BLAST sets this variable to `OFF` (see “Command Line Switches” on page 10).

NOTE: If `@KEYBOARD` is set to `ON`, it returns the value `1`; if it is set to `OFF`, it returns the value `0`.

10.7x

@KEYFILE

read/write
filename

Specifies a user-defined keyboard map for a particular keyboard or application. Keyboard maps are created with blastkbd, the BLAST keyboard remapping utility (see “Keyboard Mapping Utility for 10.7x” on page 315).

Setup field: Keyboard File

@KFILETYP

read/write
TEXT **[BINARY]**

For Kermit transfers, specifies the type of file being transferred.

Kermit Protocol subwindow: Transfer Type

@KFNAMCONV

read/write
[YES] NO

For Kermit transfers, converts a filename from local format to common format.

Kermit Protocol subwindow: Filename Conversion

@KREOPKT

read/write
^A – ^Z **[^M]**

For Kermit transfers, specifies a control character to terminate each packet received. The same control character must also be used by the remote Kermit.

Kermit Protocol subwindow: End-of-Packet Char

@KRPADCH

read/write
^A – ^Z **[^@]**

For Kermit transfers, specifies an alternate character to pad each packet received.

Kermit Protocol subwindow: Pad Character

@KRPADDNG

read/write
[0] – 99

For Kermit transfers, specifies the number of padding characters to request per packet.

Kermit Protocol subwindow: Padding

@KRPKTLEN

read/write
10 – 2000 **[90]**

For Kermit transfers, specifies the packet size your system will use when it receives a file. Note that the remote Kermit's Send packet size should also be set to this length.

Kermit Protocol subwindow: Packet Size

@KRSOPKT

read/write
[^A] – ^Z

For Kermit transfers, specifies the control character that marks the start of each packet received by your system. The same control character must also be used by the remote Kermit.

Kermit Protocol subwindow: Start-of-Packet Char

@KRTIMEOUT

read/write
0 – 99 **[10]**

For Kermit transfers, specifies the number of seconds that the computer will wait to receive a packet before requesting that it be resent.

Kermit Protocol subwindow: Timeout

@KSAVEINC

read/write
[DISCARD] KEEP

For Kermit transfers, specifies whether to `KEEP` or `DISCARD` files not completely received, such as a file being transferred when you abort a Get command.

Kermit Protocol subwindow: Incomplete File

@KSEOPKT

read/write
^A – ^Z **[^M]**

For Kermit transfers, specifies a control character to terminate each packet sent by your system. The same control character must also be used by the remote Kermit.

Kermit Protocol subwindow: End-of-Packet Char

@KSPADCH

read/write
^A – ^Z **[^@]**

For Kermit transfers, specifies an alternate character to pad each packet sent by your system.

Kermit Protocol subwindow: Pad Character

@KSPADDNG

read/write
[0] – 99

For Kermit transfers, specifies the number of padding characters to send per packet.

Kermit Protocol subwindow: Padding

@KSPKTLEN

read/write
10 – 2000 **[90]**

For Kermit transfers, specifies the packet size your system will use when it sends a file. Note that the packet size of the remote Kermit must also be set to this length.

Kermit Protocol subwindow: Packet Size

@KSSOPKT

read/write
[^A] – ^Z

For Kermit transfers, specifies the control character that marks the start of each packet sent by your system. The same control character must also be used by the remote Kermit.

Kermit Protocol subwindow: Start-of-Packet Char

@KWARNING

read/write
[ON] OFF

For Kermit transfers, specifies whether Kermit will automatically rename a received file if another file with the same name already exists in the current directory. If `@KWARNING` is set to `ON`, Kermit automatically renames the file by adding a number (0001, 0002, etc.) to the filename; if it set to `OFF`, Kermit overwrites the file.

Kermit Protocol subwindow: Warning

@LAUNCHST

read/write
any ASCII string **[\r]**

For BLAST protocol transfers, specifies the launch string to be appended to BLAST protocol blocks. Any ASCII string may be used, with control characters represented by a backslash followed by a three-digit octal number (see the discussion of special control characters on page 221). The default is a carriage return (\r). This variable may be necessary for protocol converter connections.

BLAST Protocol subwindow: Launch String

@LINEDLY

read/write
[0] – 999

Specifies the length of time (in tenths of a second) that BLAST pauses after sending a line of characters and a carriage return during a text upload.

Setup field: Line Delay

@LOCECHO

read/write
YES **[NO]**

Specifies whether BLAST will echo typed characters to the screen while in Terminal mode. If `@LOCECHO` is set to `YES`, BLAST will display typed characters before sending them out the communication port; if `@LOCECHO` is set to `NO`, the characters will be displayed only if the remote computer sends them back.

If `@LOCECHO` is set to `YES` and double characters are displayed on the screen, change the setting to `NO`.

Setup field: Local Echo

@LOGDATEFORMAT

read/write
template

Sets the format of the date written in the date stamp of the log file. Setting `@LOGDATEFORMAT` overrides the format in which BLAST was started. The format of dates written in the log file will be determined by the template set by the user. The value of the replacement sequences are the same as those described above in the `@DATEFORMAT` reserved variable.

@LOGFILE

read/write
filename

Stores the name of the log file that will record all communications session activity. Setting `@LOGFILE = @LOGFILE` flushes the log file buffers to disk. Setting `@LOGFILE = ""` closes the current log file.

Setup field: Log File

@LOGTIMEFORMAT

read/write
template

Sets the format of the time written in the time stamp of the log file. Setting `@LOGTIMEFORMAT` overrides the format in which BLAST was started.The format of times written in the log file will be determined by `@LOGTIMEFORMAT` template set by the user. The value of the replacement sequences are the same as those described above in the `@DATEFORMAT` reserved variable.

@LOGTIMO

read/write
0 – 999 **[120]**

For BLAST protocol, specifies the number of seconds that BLAST will attempt to establish a filetransfer session with the remote computer before aborting. Logon Timeout affects BLAST protocol Filetransfer and Access modes. If zero is entered, no timeout will occur and BLAST will attempt to establish a filetransfer session with the remote computer indefinitely.

BLAST Protocol subwindow: Logon T/O

@MODEM

read/write
any valid modem type

Stores the modem type on the local computer. The name must be defined in the modems.scr library or exist as a separate script.

Setup field: Modem Type

@NUMDISC

read/write
0 – 9 **[3]**

For BLAST protocol, sets the number of additional disconnect blocks (after the first disconnect block) that BLAST sends when exiting Filetransfer mode. Possible values are `0`–`9`. The default value of `3` indicates four total disconnect blocks.

BLAST Protocol subwindow: Number of Disconnect Blocks

@ONERROR

read/write
[STOP] CONTINUE

Specifies BLAST's response to nonfatal BLASTscript errors. A nonfatal error is one that results in the message "Press any key to continue."

When `@ONERROR` is set to `STOP`, BLAST will pause when an error is encountered, display the appropriate message, and wait for the user to press a key before continuing. When `@ONERROR` is set to `CONTINUE`, BLAST will display the same message, pause for one second, and then automatically continue script execution.

@ORGANS

read/write
[ORIGINATE] ANSWER

Specifies how the Connect command will operate. If `@ORGANS` is set to `ANSWER`, Connect will wait for a remote computer to establish the communications link. If it is set to `ORIGINATE`, Connect will try to dial a number.

Setup field: Originate/Answer

@PAKTSZ

read/write
1 – 4085 **[256]**

For BLAST protocol transfers, specifies the size of the packet.

Setup field: Packet Size

@PARITY

read/write
[NONE] ODD EVEN

Sets the device driver parity of the serial port. This setting should match that of the remote system

Setup field: Parity

@PASSWORD

write-only
user-defined

Stores the user's password for the remote computer. The systems.scr library program uses `@PASSWORD` to answer prompts from a multi-user computer. The `CONNECT` command will prompt the user to enter a password if none is specified in the Setup. Thereafter, the variable `@PASSWORD` contains the value entered by the user. For security, the value of `@PASSWORD` cannot be displayed to the screen. This feature applies to all string values that match `@PASSWORD`. Thus, script commands such as

```
set @trick = @PASSWORD
display @trick
```

will *not* display the value of the password.

BLAST makes an effort to keep stored passwords secure. Unfortunately, it is a very simple task to echo a stored password off either a modem or a remote system that has echo enabled. A script as simple as "`tsend @password`" can compromise stored passwords. If the security of a password is vital, BLAST recommends not storing it in the setup. If a password must be stored in the setup, you should take other measures to keep the setup secure. For more information on security, consult your system documentation and Chapter 11.

Setup field: Password

@PHONENO

read/write
user-defined

Specifies the phone number of the remote computer. The `CONNECT` statement uses this number to dial out.

Setup field: Phone Number

@PROMPTCH

read/write
[NONE] any ASCII character

Defines the prompt character used during text uploads to half-duplex systems. BLAST waits after each line for the remote computer to send the prompt before sending the next line.

Setup field: Prompt Char

@PROTOCOL

read/write
[BLAST] KERMIT
XMODEM XMODEM1K
YMODEM YMODEM G ZMODEM

10.8x	FTP

Specifies the protocol for a communications session.

Setup field: Protocol

@RBTOT

read-only

If Extended Logging is enabled, holds the total number of bytes received during the file transfer session. You must write a display statement (e.g. `Display "@RBTOT is ", @RBTOT`) for this variable to be displayed in the Extended Log file. See the description of `@XLOG` for more information.

@RBYTES

read-only

In the BLAST Extended Log, holds the number of bytes received in the current transfer. Note that this value can be different than the actual file size. You must have Extended Logging enabled for this variable to return a value. See `@XLOG` for more information.

@RCLASS

read-only

For BLAST protocol, stores the BLAST class number of the remote system. This is valid during and after file transfer.

@RCOMP_LEV

read/write
0 – 6 **[4]**

For BLAST protocol transfers, specifies the maximum receiving level of compression that can be used during a session. Level `0` specifies no compression; level `6` specifies the highest compression level.

BLAST Protocol subwindow: Receive Compression Level

@RETRAN

read/write
0 – 9999 **[4]**

For BLAST protocol transfers, sets the maximum number of seconds BLAST will pause before resending a packet. For example, if

@WDWSIZ is set to 5 and @RETRAN is set to 30, BLAST will attempt to resend the fifth packet every 30 seconds if no acknowledgement is received.

BLAST Protocol subwindow: Retransmit Timer

@RFAILURE read-only

For BLAST protocol, stores the number of files unsuccessfully received during a file transfer session.

@RLINEQ read-only

For BLAST protocol transfers, stores the current receiving line quality. Possible values are GOOD, FAIR, POOR, or DEAD.

@RLQ read-only

In the BLAST Extended Log, holds the line quality for the file being received. You must have Extended Logging enabled for this variable to return a value. Possible values are GOOD, FAIR, POOR, or DEAD. See the description of @XLOG for more information.

@RNAME read-only

In the BLAST Extended Log, holds the name of the file being received. You must have Extended Logging enabled for this variable to return a value. See the description of @XLOG for more information.

@ROPTIONS read-only

In the BLAST Extended Log, holds the value of the options for the file being received. You must have Extended Logging enabled for this variable to return a value. See the description of @XLOG for more information.

@RPACK read-only

In the BLAST Extended Log, holds the number of packets received in the transfer. You must have Extended Logging enabled for this variable to return a value. See the description of @XLOG for more information.

@RPTOT read-only

In the BLAST Extended Log, holds the total number of packets received during the file transfer session. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RRET read-only

In the BLAST Extended Log, holds the number of retries for the file being received. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RRTOT read-only

In the BLAST Extended Log, holds the total number of retries for files being received during the file transfer session. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RSERIAL read-only

For BLAST protocol, stores the serial number of the BLAST version running on the remote system.

@RSITE read-only

For BLAST protocol, stores the BLAST site number of the remote system. This is valid during and after file transfer.

@RSIZE read-only

In the BLAST Extended Log, holds the size of the file being received. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RSTART read-only

In the BLAST Extended Log, holds the interrupt start point for an interrupted received file. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RSTATUS read-only

In the BLAST Extended Log, holds the completion status of the file being received. Possible values are:

`RCOMP` – Receive completed.

`LERROR` – Receive not completed, due to local error.

`RERROR` – Receive not completed, due to remote error.

`RINTR` – Receive not completed, due to operator interruption.

You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RSUCCESS read-only

For BLAST protocol, stores the number of files successfully received during a file transfer session.

@RTIME read-only

In the BLAST Extended Log, holds the elapsed time for the file being received. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@RTSCTS read/write

[YES] NO

Specifies whether hardware flow control is enabled. Not all computers support RTS/CTS flow control. The value of this variable is valid only if the serial port device driver returns the correct code.

Setup field: RTS/CTS Pacing

@SBTOT read-only

If Extended Logging is enabled, holds the total number of bytes sent during the file transfer session. You must write a display statement (e.g. `Display "@SBTOT is ", @SBTOT`) for this variable to be displayed in the Extended Log file. See the description of `@XLOG` for more information.

@SBYTES read-only

In the BLAST Extended Log, holds the number of bytes sent in the current transfer. Note that this value can be different than the actual file size. You must have Extended Logging enabled for this variable to return a value. See `@XLOG` for more information.

@SCOMP_LEV read/write
0 – 6 **[4]**

For BLAST protocol transfers, specifies the maximum sending compression level that can be used during a session. Level `0` specifies no compression; level `6` specifies the highest compression level.

BLAST Protocol subwindow: Send Compression Level

@SCRFILE read/write
filename

Specifies the name of a BLAST script that will start immediately after BLAST begins execution.

Setup field: Script File

@SCRIPTERR read/write
any integer

Returns the numeric value of the last error that occurred in the BLAST script.

@SCRLREG read/write
[ON] OFF

Controls data display in the scrolling region (lines 5–24). If `@SCRLREG` is set to `ON`, characters received in Terminal mode will be displayed and BLAST scripts can use the `DISPLAY` statement. If BLAST is started in video-suppress mode (`-n` switch on the operating system command line), `@SCRLREG` is set to `OFF` (see “Command Line Switches” on page 10).

NOTE: If `@SCRLREG` is set to `ON`, it returns the value `1`; if it is set to `OFF`, it returns the value `0`.

@SERIAL read-only

Stores the serial number of the BLAST version running on the local system.

@SETUPDIR read-only

Specifies the directory path in which BLAST setup files are stored, as specified in the SETUPDIR environment variable (see "Environment Variables" on page 7).

@SFAILURE read-only

For BLAST protocol, stores the number of files unsuccessfully sent during a file transfer session.

@SITE read-only

Stores the BLAST site number of the local system.

@SLINEQ read-only

For BLAST protocol, stores the current sending line quality during a file transfer. Increase packet size to take advantage of clean lines, or decrease packet size to avoid problems with noisy lines. Possible values are `GOOD`, `FAIR`, `POOR`, or `DEAD`.

@SLQ read-only

In the BLAST Extended Log, holds the line quality for the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SNAME read-only

In the BLAST Extended Log, holds the name of the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SOPTIONS read-only

In the BLAST Extended Log, holds the value of the options for the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SPACK read-only

In the BLAST Extended Log, holds the number of packets sent in the transfer. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SPTOT read-only

In the BLAST Extended Log, holds the total number of packets sent during the file transfer session. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SRET read-only

In the BLAST Extended Log, holds the number of retries for the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SRTOT read-only

In the BLAST Extended Log, holds the total number of retries for files being sent during the file transfer session. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SSIZE read-only

In the BLAST Extended Log, holds the size of the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SSTART read-only

In the BLAST Extended Log, holds the interrupt start point for an interrupted sent file. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@SSTATUS read-only

In the BLAST Extended Log, holds the completion status of the file being sent. Possible values are:

SCOMP – Send completed.

LERROR – Send not completed, due to local error.

RERROR – Send not completed, due to remote error.

SINTR – Send not completed, due to operator interruption.

You must have Extended Logging enabled for this variable to return a value. See the description of @XLOG for more information.

@SSUCESS read-only

For BLAST protocol, stores the number of files successfully sent during a file transfer session.

@STATUS read/write command-specific

Returns a condition code set by the last statement that reported a completion status. Most statements that succeed set @STATUS to 0 and return a nonzero value for an error. For example, the FILETRANSFER command sets @STATUS to 0 if Filetransfer mode was successfully entered. @STATUS does *not,* however, reflect the success of an entire FILETRANSFER *block,* but rather the @STATUS setting of the last command in the block capable of setting @STATUS. (To check the overall success of a FILETRANSFER block, use the reserved variable @EFERROR).

Some commands that return numeric results (e.g., STRINX, TTRAP) set @STATUS to 0 to indicate a null condition.

On returning from a called script, @STATUS is set to the numeric constant given in the RETURN statement, or to 0 if no numeric constant is given.

For a list of commands that set @STATUS, see “Commands That Set @STATUS” on page 223.

@STIME read-only

In the BLAST Extended Log, holds the elapsed time for the file being sent. You must have Extended Logging enabled for this variable to return a value. See the description of @XLOG for more information.

10.8x	**@SYMBOLTYPE** read-only Returns the results of the last SYMTYPE command—NONE, BINARY, or STRING.

@SYSDESC

read/write
user-defined

Stores a user-defined description of the remote computer. This field may be up to 40 characters. No special processing is done based on the information in this field.

Setup field: Description

@SYSTYPE

read/write
any valid system type

Specifies the remote computer type (UNIX, VMS, etc.). The systems.scr library uses this variable to determine how to perform certain system functions, such as logging on and disconnecting from remote multi-user computers.

Setup field: System Type

@TIME

read-only

Contains the current time in *hh:mm:ss* format. This is a read-only variable; an error message will be displayed if a script attempts to write to it.

@TIMEFORMAT

read/write
template

Sets the format of the @TIME variable. Setting the @TIMEFORMAT reserved variable overrides the format in which BLAST was started. The format of the output of the @TIME reserved variable will be determined by the template set by the user. The value of the replacement sequences are the same as those described above in the @DATEFORMAT reserved variable.

@TRANSTAT

read/write
[ON] OFF

Controls the display of the File Transfer Status Area. If @TRANSTAT is set to ON, the area is active. This variable is set to OFF when

BLAST is started in video-suppress mode (`-n` on the operating system command line; see “Command Line Switches” on page 10).

NOTE: If `@TRANSTAT` is set to `ON`, it returns the value `1`; if it is set to `OFF`, it returns the value `0`.

10.8x	**@TRAPCNT** read-only Returns the number of bytes read from the last `TTRAP`. `TTRAP` must be preceded by `SETTRAP`.

@TRPASSWD

write-only
up to 8 characters

For BLAST protocol, stores a password that a remote user must send before a file transfer is allowed. If this variable is set to other than null, then the remote computer must send the password before a file can be transferred to or from your computer.

NOTE: `@TRPASSWD` is intended to validate remote users logging onto your system. If the BLAST running on the local system executes a script that sets `@TRPASSWORD` to something other than a null, the local computer will not be able to receive files without the remote computer sending the password.

BLAST Protocol subwindow: Transfer Password

@TTIME

read-only

In the BLAST Extended Log, holds the total elapsed time of the file transfer session. You must have Extended Logging enabled for this variable to return a value. See the description of `@XLOG` for more information.

@USERID

read/write
user-defined

Stores the user’s identification for the remote computer. The systems.scr library uses this variable in answering the logon prompts from a multi-user computer.

Setup field: Userid

@USERIF

read/write
[ON] OFF

Controls data display in the menu region (lines 1–4). If @USERIF is set to ON, the menu region is displayed; if it is set to OFF, lines 1–4 become part of the scrolling region. When BLAST is started in the video-suppress mode (-n on the operating system command line), this variable is turned OFF (see "Command Line Switches" on page 10).

NOTE: If @USERIF is set to ON, it returns the value 1; if it is set to OFF, it returns the value 0.

@VERSION

read-only

Stores the version of BLAST that is running.

10.7x

@VT8BIT

read/write
[7] 8

For VT220 and VT320 emulation, specifies whether "C1" control characters are represented in the 8-bit environment or as 7-bit escape sequences.

VT Emulation subwindows: 7/8 Bit Controls

@VTANSBACK

read/write
user-defined ASCII string

For VT emulation, contains a message to be sent to the remote computer upon receiving an inquiry (^E). The field can be up to 30 characters in length. The default value is an empty string—nothing is sent.

VT Emulation subwindows: Answerback Msg

@VTAUTOWRAP

read/write
YES **[NO]**

For VT emulation, specifies whether text typed at the right margin will automatically wrap to the next line.

VT Emulation subwindows: Auto Wrap

@VTCLRSCRN read/write

YES **[NO]**

For VT emulation, clears the terminal's video display. Setting `@VTCLRSCRN` to `YES` clears the display; the value is then reset to `NO`.

VT Emulation subwindows: Clear Screen

@VTCURSOR read/write

[NORMAL] APPLICATION

For VT emulation, specifies whether the cursor keys will control cursor movement or send application control functions.

VT Emulation subwindows: Cursor Keys Mode

@VTCURSTYPE read/write

BLOCK **[LINE]**

10.7x For VT100/52 and VT220 emulation, specifies whether the cursor is displayed as a reverse-video block or as an underline character.

VT Emulation subwindows: Cursor Type

@VTDISP132 read/write

[80] 132

For VT emulation, specifies column display for text.

VT Emulation subwindows: 80/132 Columns

@VTHSCROLL **[JUMP]** SMOOTH NONE

For VT emulation, specifies how to scroll data on an 80-column display when the emulator is in 132-column mode. `SMOOTH` scroll will change the view of the display only as necessary to display the cursor position. `JUMP` scroll will adjust the view by showing either the first 80 columns or the last 80 columns. When `NONE` is selected, the display will not scroll and the cursor may disappear from view. This value is ignored if `@VTCOMPRESSED` is set to `YES`.

VT Emulation subwindows: Horizontal Scroll

@VTHSCROLLN 1 – 53 **[10]**

For VT emulation, specifies the number of columns to move when the Scroll Left or Scroll Right keys are pressed. The value is used when Jump Scroll has been selected as the scroll mode.

VT Emulation subwindows: Jump Scroll Inc

@VTINTL **[USASCII]** UK FRENCH GERMAN ITALIAN SPANISH DANISH

For VT220 and VT320 emulation, specifies whether 7- or 8-bit data is used for international support. The default value, `USASCII`, allows 8-bit data with the high-order data used for international characters.

VT Emulation subwindows: Intl Char Set

@VTKEYPAD **[NUMERIC]** APPLICATION

10.7x For VT emulation, specifies whether the numeric keypad keys will send numbers or programming functions defined by the application.

VT Emulation subwindows: Keypad Mode

@VTNEWLINE read/write YES **[NO]**

For VT emulation, selects whether the ENTER key will move the cursor to a new line.

VT Emulation subwindows: New Line

@VTPRINT read/write **[NORMAL]** AUTO CONTROLLER

For VT emulation, specifies when information is sent to the printer. In `AUTO` print mode, each line of received text is displayed and printed; in `CONTROLLER` mode, all received data is sent directly to the printer without displaying it on the screen; in `NORMAL` mode, the user initiates printing from the keyboard.

VT Emulation subwindows: Print Mode

@VTPRINTPAGE read/write
[SCROLL REGION]
FULL PAGE

For VT emulation, specifies how much of the screen to print when you press the PRINT SCREEN key.

VT Emulation subwindows: Print Screen

@VTRESET read/write
YES **[NO]**

For VT emulation, specifies whether many of the VT operating features are reset to their factory default values. If `@VTRESET` is set to `YES`, the features are reset; the value of this variable is then authomatically reset to `NO`.

VT Emulation subwindows: Reset Terminal

10.7x **@VTTEXTCURS** read/write
[YES] NO

For VT emulation, specifies whether to display the text cursor.

VT Emulation subwindows: Text Cursor

@VTUSERCHAR read/write
[DEC SUPPLEMENTAL]
ISO LATIN-1

For VT320 emulation, specifies either `DEC SUPPLEMENTAL` or `ISO LATIN-1` character set as the user preferred character set.

VT Emulation subwindows: User Pref Char Set

@VTUSERKEYS **read/write**
[UNLOCKED] LOCKED

For VT200 and VT320 emulation, selects whether the host system can change user-defined key definitions.

VT Emulation subwindows: User Def Keys.

@WDWSIZ

read/write
1 – **[16]**

For BLAST protocol, specifies the window size of the "B" protocol. "Window" refers to the number of BLAST protocol packets that can be sent to the remote without BLAST waiting for an acknowledgement from the remote. As packets are acknowledged, the start point of the window is adjusted, or "slides." See "BLAST Protocol Design" on page 101 for a fuller discussion of window size.

BLAST Protocol subwindow: Window Size

@WT4ECHO

read/write
YES **[NO]**

Specifies whether BLAST will wait for the remote computer to echo each character of uploaded text before sending the next character.

Setup field: Wait For Echo

10.7x

@WYANSBACK

read/write
user-defined

For WYSE emulation, contains a user-created message to be sent to the host when an inquiry is received.

WYSE Emulation subwindow: Answerback

@WYAUTOPAGE

read/write
YES **[NO]**

For WYSE emulation, specifies whether the cursor can move off the current page when an attempt is made to move the cursor before the home position or beyond the end of the page.

WYSE Emulation subwindow: Auto Page

@WYAUTOSCROLL

read-only
[YES]

For WYSE emulation, specifies scrolling of the terminal display when the cursor reaches the bottom of a page. This field is read-only and cannot be changed.

WYSE Emulation subwindow: Auto Scroll

	@WYAUTOWRAP read/write **[YES]** NO For WYSE emulation, specifies whether a new line is automatically performed when a character is placed in the last column of a row (column 80 or 132). *WYSE Emulation subwindow: Auto Wrap*
	@WYBLOCKEND read/write **[US/CR]** CRLF/ETX For WYSE emulation, specifies which characters are used to mark the end-of-line and end-of-block when the terminal is in block mode. *WYSE Emulation subwindow: Block End*
10.7x	**@WYCOMMODE** **[CHARACTER]** BLOCK For Wyse emulation, specifies whether data is sent after each keystroke (character mode) or packaged into blocks. *WYSE Emulation subwindow: Comm Mode*
	@WYDISP80 read/write **[80]** 132 For WYSE emulation, specifies a display of `80` or `132` columns per row. *WYSE Emulation subwindow: Columns*
	@WYDSPCURSOR read-only **[YES]** For WYSE emulation, specifies that the cursor is visible. This field is read-only and cannot be changed. *WYSE Emulation subwindow: Display Cursor*

10.7x

@WYENTER

read/write
[CR] CRLF TAB

For WYSE emulation, specifies the character to send when the keypad ENTER key is pressed.

WYSE Emulation subwindow: Enter

@WYEXPNDMEM

read/write
YES **[NO]**

For WYSE emulation, specifies expanded memory use.

WYSE Emulation subwindow: Expanded Memory

@WYPAGELEN

read/write
[1*DATA LINES]
2*DATA LINES 4*DATA LINES

For WYSE emulation, specifies the length of a screen page.

WYSE Emulation subwindow: Page Length

@WYRETURN

read/write
[CR] CRLF TAB

For WYSE emulation, specifies the character to send when the RETURN key is pressed.

WYSE Emulation subwindow: Return

@WYSCROLLINC

read/write
1– 53 **[10]**

For Wyse emulation, specifies the scroll increment. This value is used when 132 columns per row has been selected and compressed display is not utilized.

Wyse Emulation subwindow: Horiz Scroll Inc

10.7x	**@WYSEWORD** read/write YES **[NO]** For WYSE emulation, specifies whether keys send Wordstar™ functions instead of standard key codes. The only keys affected are WYSE keys that can be remapped with the blastkbd utility (see “Keyboard Mapping Utility for 10.7x” on page 315). *WYSE Emulation subwindow: Wyseword* **@WYWRITEPROT** read/write **[DIM]** REVERSE NORMAL For WYSE emulation, specifies the attribute used to display protected fields. *WYSE Emulation subwindow: Write Protect*
10.8x	**@XCRC** read/write **[CRC]** CHECKSUM For Xmodem transfers, specifies whether the error detection is `CRC` or `CHECKSUM`. *Setup field: Error Detection*

@XLOG

read/write
ON **[OFF]**

Enables Extended Logging, which provides detailed information about BLAST protocol file transfers. Extended Log values may be read from the variables listed below. When Extended Logging is enabled, all the values below are listed in the log file except for `@RBTOT` and `@SBTOT`, which may be written to the log file by issuing a display statement (e.g. `display "@RBTOT is ", @RBTOT`).

```
@SNAME       @RNAME       @STIME     @RTIME
@SOPTIONS    @ROPTIONS    @SPACK     @RPACK
@SSTATUS     @RSTATUS     @SRET      @RRET
@SSIZE       @RSIZE       @SPTOT     @RPTOT
@SSTART      @RSTART      @SRTOT     @RRTOT
@SBYTES      @RBYTES      @SBTOT     @RBTOT
@SLQ         @RLQ         @TTIME
```

Extended Logging may also be enabled with the `-x` command line switch (see “Command Line Switches” on page 10).

@XLTFILE

read/write
filename

Stores the name of the Translate File used in Terminal mode to filter, translate, or substitute characters (see “Translate File Format” on page 306).

Setup field: Translate File

@XONXOFF

read/write
YES **[NO]**

Specifies whether software flow control is enabled. Not all computers support XON/XOFF flow control.

Setup field: XON/XOFF Pacing

10.8x

@XPADC

read/write
any character in decimal **[00]**

For Xmodem transfers, specifies the pad character. This parameter may also be set from the command line with `-p` command line switch (“Command Line Switches” on page 10).

XYmodem Protocol subwindow: Pad character

@XYCONVR

read/write
ASCII **[BINARY]**

For Xmodem and Ymodem transfers, specifies whether received files will be treated as `ASCII` or `BINARY`.

XYmodem Protocol subwindow: File Conversion

@XYCONVS

read/write
ASCII **[BINARY]**

For Xmodem and Ymodem transfers, specifies whether files sent will be treated as `ASCII` or `BINARY`.

XYmodem Protocol subwindow: File Conversion

@XYEOT

read/write
10 – 6000 **[100]**

For Xmodem and Ymodem transfers, specifies EOT (end-of-transmission) timeout in hundredths of a second.

EOT timeout for Xmodem and Ymodem may also be specified with the `-e` command line switch (see “Command Line Switches” on page 10)

XYmodem Protocol subwindow: EOT Timeout

@XYRLTR

read/write
CR **[CR/LF]**

For Xmodem and Ymodem transfers, specifies how line termination is treated if `@XYCONVS` is set to `ASCII`.

`CR` – for files received, replaces all carriage returns (`CR`) with linefeeds (`LF`); e.g., for ASCII files received from Macintosh platform.

10.8x

`CR/LF` – for files received, deletes any carriage return (`CR`) *that is followed by a line feed* (`LF`); e.g., for ASCII files sent to DOS and Windows platforms.

XYmodem Protocol subwindow: Remote Line Termination

@XYRLTS

read/write
CR **[CR/LF]**

For Xmodem and Ymodem transfers, specifies how line termination is treated if `@XYCONVS` is set to `ASCII`.

`CR` – for files sent, replaces line feeds (`LF`) with carriage returns (CR); e.g., for ASCII files sent to Macintosh platform.

`CR/LF` – for files sent, adds a carriage return (`CR`) before a line feed (`LF`); e.g., for ASCII files sent to DOS and Windows platforms.

XYmodem Protocol subwindow: Remote Line Termination

10.8x	**@ZMALT** read/write **[CR/LF]** LF For sending ASCII files to nonstandard implementations of Zmodem, specifies line-feed conversion for ASCII files. When `@ZMCONVS = "ASCII",` the default `CR/LF` specifies that line feeds be converted to `CR/LF`; `LF` specifies no conversion. *Zmodem Protocol subwindow: ASCII Line Termination*

@ZMAUTODOWN

read/write
YES **[NO]**

For Zmodem transfers, specifies Auto Receive mode, which begins downloading immediately after entering Filetransfer mode.

Zmodem Protocol subwindow: Auto Receive

@ZMBLKLN

read/write
[0] 24 – 1024

For Zmodem transfers, overrides the default block length, which is determined by the baud rate of the connection. The default, `0`, specifies no limit to block length.

Zmodem Protocol subwindow: Limit Block Length

@ZMCONVR

read/write
[ASCII] BINARY

For Zmodem transfers, specifies whether received files will be treated as `ASCII` or `BINARY`. For correct file conversion to ASCII, the remote computer must send the files as ASCII.

Zmodem Protocol subwindow: File Conversion

@ZMCONVS

read/write
[NONE] ASCII BINARY

For Zmodem transfers, specifies whether files sent are to be treated as `BINARY` or `ASCII`, overriding the File Conversion setting of the receiving system. `NONE` specifies no override.

Zmodem Protocol subwindow: Conversion Override

@ZMCRC

read/write
16 BITS **[32 BITS]**

For Zmodem transfers, specifies which CRC error-detection is to be used.

Zmodem Protocol subwindow: CRC

@ZMCTLESCR

read/write
YES **[NO]**

For Zmodem transfers, specifies whether all control characters received will be link-escape encoded for transparency.

Zmodem Protocol subwindow: Esc All Control Chars

@ZMCTLESCS

read/write
YES **[NO]**

For Zmodem transfers, specifies whether all control characters sent will be link-escape encoded for transparency.

Zmodem Protocol subwindow: Esc All Control Chars

@ZMEXIST

read/write
YES **[NO]**

For Zmodem transfers, specifies whether transfers will occur only if the file already exists on the destination system.

Zmodem Protocol subwindow: File Must Already Exist

@ZMFRMLEN

read/write
[0] 24 – 1024

For Zmodem transfers, limits frame length and forces the sender to wait for a response from the receiver before sending the next frame. The default, `0`, specifies no limit to frame length.

Zmodem Protocol subwindow: Limit Frame Length

@ZMMANAGR

read/write
NONE PROTECT
[CLOBBER] APPEND

For Zmodem transfers, specifies a file management option for files received. See the File Management setup field on page 97 for a description of each option.

Zmodem Protocol subwindow: File Management

@ZMMANAGS

read/write
[NONE] PROTECT
CLOBBER NEWER
NEWER/LONGER
DIFFERENT APPEND

For Zmodem transfers, specifies a file management option for files sent. See the Management Option setup field on page 95 for a description of each option.

Zmodem Protocol subwindow: Management Option

@ZMRESUME

read/write
YES **[NO]**

For Zmodem transfers, specifies continuation of an aborted file transfer from point of interruption. The destination file must already exist and must be smaller than the source file.

Zmodem Protocol subwindow: Resume Interrupted File

@ZMWINDOW

read/write
[0] – 9999

For Zmodem transfers, specifies the size of the transmit window. The default, `0`, specifies no limit to the size of the transmit window.

Zmodem Protocol subwindow: Size of Tx Window

Chapter 17

Data Stream Control

Introduction

All versions of BLAST support data filtering and translation of incoming and outgoing data streams. This chapter describes these features as well as the standard BLAST terminals, TTY and PASSTHRU. In addition, this chapter describes terminal emulation and keyboard mapping, which are available with BLAST Professional UNIX 10.7x. Through terminal emulation, BLAST provides terminal functionality for a range of popular character terminals. With keyboard mapping, you can reassign the functions of the standard keyboard keys as well as the “BLAST keys” that control BLAST functions.

Data Stream Filtering and Alteration

BLAST allows for the translation, substitution, or filtering (removal) of individual characters in the data stream during terminal sessions. This character manipulation can be used to:

◊ Prevent the display of unwanted characters.

◊ Display international character sets.

◊ Prevent the transmission of certain key codes.

◊ Remap keys to send characters other than their defaults.

◊ Prevent characters from being saved in the capture file.

◊ Prevent characters from being sent with a file upload.

For example, Dow Jones News Service sends special start- and end-of-record characters that print non-ASCII characters on the screen. The standard translate file supplied with BLAST filters out these characters so that they do not appear on your display. If you wanted to automate your access to Dow Jones by writing a script, you might need to TTRAP for these filtered characters. For the TTRAP to see them, you would have to change the filter in order to allow these characters to pass.

Translate File Format

A copy of the standard translate file is on your distribution media as "translat.tbl." This file is distributed with the defaults used when the Translate File setup field (page 73) is empty. The BLAST translate file contains two tables: the receive table, which operates on characters received from the remote system, and the transmit table, which operates on characters sent to the remote system.

The receive and transmit tables within a BLAST translate file contain an array of 256 hexadecimal values. These values correspond to the 8-bit ASCII character set. The decimal value of a character ranging from 0 to 255 is used as an index to the character positions in the table. The hexadecimal value at that location in the table is substituted for the hexadecimal value of the original character.

Translat.tbl contains the following receive and transmit default tables:

```
:RECVTABL
     -00,    -01,    -02,    -03,    -04,    -05,    -06,    07,
      08,     09,     0A,     0B,     0C,     0D,     0E,    0F,
     -10,    -11,    -12,    -13,    -14,    -15,    -16,   -17,
     -18,    -19,    -1A,    -1B,    -1C,    -1D,    -1E,   -1F,
      20,     21,     22,     23,     24,     25,     26,    27,
      28,     29,     2A,     2B,     2C,     2D,     2E,    2F,
      30,     31,     32,     33,     34,     35,     36,    37,
      38,     39,     3A,     3B,     3C,     3D,     3E,    3F,
      40,     41,     42,     43,     44,     45,     46,    47,
```

```
      48,    49,    4A,    4B,    4C,    4D,    4E,    4F,
      50,    51,    52,    53,    54,    55,    56,    57,
      58,    59,    5A,    5B,    5C,    5D,    5E,    5F,
      60,    61,    62,    63,    64,    65,    66,    67,
      68,    69,    6A,    6B,    6C,    6D,    6E,    6F,
      70,    71,    72,    73,    74,    75,    76,    77,
      78,    79,    7A,    7B,    7C,    7D,    7E,    7F,
     -00,   -01,   -02,   -03,   -04,   -05,   -06,    07,
      08,    09,    0A,    0B,    0C,    0D,    0E,    0F,
     -10,   -11,   -12,   -13,   -14,   -15,   -16,   -17,
     -18,   -19,   -1A,    1B,   -1C,   -1D,   -1E,   -1F,
      20,    21,    22,    23,    24,    25,    26,    27,
      28,    29,    2A,    2B,    2C,    2D,    2E,    2F,
      30,    31,    32,    33,    34,    35,    36,    37,
      38,    39,    3A,    3B,    3C,    3D,    3E,    3F,
      40,    41,    42,    43,    44,    45,    46,    47,
      48,    49,    4A,    4B,    4C,    4D,    4E,    4F,
      50,    51,    52,    53,    54,    55,    56,    57,
      58,    59,    5A,    5B,    5C,    5D,    5E,    5F,
      60,    61,    62,    63,    64,    65,    66,    67,
      68,    69,    6A,    6B,    6C,    6D,    6E,    6F,
      70,    71,    72,    73,    74,    75,    76,    77,
      78,    79,    7A,    7B,    7C,    7D,    7E,    7F,

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

Translat.tbl can either filter, translate, or substitute characters.

Filtering – The default values of the receive table cause it to filter the following characters (next page):

NUL	(00)	ACK	(06)	NAK	(15)	ESC	(1B)
SOH	(01)	DLE	(10)	SYN	(16)	FS	(1C)
STX	(02)	DC1	(11)	ETB	(17)	GS	(1D)
ETX	(03)	DC2	(12)	CAN	(18)	RS	(1E)
EOT	(04)	DC3	(13)	EM	(19)	US	(1F)
ENQ	(05)	DC4	(14)	SUB	(1A)		

Values to be filtered from the transmitting or receiving data stream are preceded by a minus sign. A minus sign indicates that the value following it is ignored.

Translation – The default receive table also translates all “high” ASCII characters (8-bit characters above 127 [decimal] or 7F [hexadecimal] in value) to “low” ASCII (7-bit) characters by stripping the 8th bit. You will notice in the :RECVTABL illustrated above that the 17th row of the table begins, as does the 1st row, with “-00” and that the lower half of the table duplicates the upper half.

Substitution – You can substitute a new hexadecimal value for any existing default value in either the receive or transmit table. For example, suppose that you want to replace all upper case “A”s from the received data stream with lower case “b”s. You would:

◊ Find the character “A” in the ASCII table in Appendix D. You will see that the decimal value of “A” is 65 whereas the hexadecimal value is 41.

◊ Now find the hexadecimal value located in the 65th position of the translate table. Begin counting at the upper left-hand corner of the table (“-00” or “00”), moving from left to right and counting down the rows. Start your count from zero, and count until you reach the 65th position. The value in the 65th position is 41, the hexadecimal value for “A”.

◊ Look in Appendix D again and determine the hexadecimal value for “b”. That value is 62.

◊ Replace the value 41 in the translate table with 62. From now on, all “A”s in the received data stream will be translated to “b”s.

NOTE: The default *transmit* table transmits all characters without filtering, translation, or substitution.

Creating and Editing a Translate File

When specifying new values for a translate file, be sure not to delete an entry in the table completely. This will cause all entries in the table to shift values. To modify the file:

◊ Make a copy of the translat.tbl file.

◊ Modify the new file using a word processor or ASCII text editor. *Save the file in text format only*.

◊ Locate the desired character position in the table and either enter a new value or place a minus in front of the existing value in the table.

◊ Save the new table where BLAST can access it. BLAST will look in the current directory first and then in BLASTDIR.

Text Translation Using a Translate File

Characters are altered as they are received from the remote system; therefore, what you see on the terminal screen or in a captured file is the altered data. Likewise, transmitted characters are altered after all other processing; the remote system receives altered instead of original data. It is sometimes necessary to perform text translation while receiving from or transmitting to a remote system when a file transfer protocol is not available. For example, a text file on a DOS machine has a carriage return (ASCII 13) and a line feed (ASCII 10) at the end of each line. A UNIX text file only has a line feed at the end of each line. The carriage return can easily be filtered from the data stream by placing a minus sign (-) in front of the 0D character in position 13 of the receive table.

Specifying a Translate File in Your Setup

To specify a translate file for use during a session, type its name in the Translate File setup field.

Standard BLAST Terminals

All versions of BLAST for UNIX provide two terminal types, TTY emulator and PASSTHRU. In addition, BLAST Professional UNIX 10.7x provides other terminal emulators, described in “Terminal Emulation with 10.7x” on page 311.

TTY

The BLAST TTY terminal emulator is a "generic terminal emulator" using the character values of the default translate file that allows characters to be sent without any translation or other special handling. Received characters are either displayed as text, filtered out, or interpreted as command sequences. For complete character transparency, use the PASSTHRU terminal, described in the next section.

Special Considerations

During TTY emulation, the following received ASCII characters are interpreted as command sequences (numeric values are in hexadecimal):

BEL	(07)	Bell
BS	(08)	Backspace
HT	(09)	Horizontal tab
LF	(0A)	Line feed
CR	(0D)	Carriage return

The TTY emulator filters the following characters:

NUL	(00)	ACK	(06)	NAK	(15)	ESC	(1B)
SOH	(01)	DLE	(10)	SYN	(16)	FS	(1C)
STX	(02)	DC1	(11)	ETB	(17)	GS	(1D)
ETX	(03)	DC2	(12)	CAN	(18)	RS	(1E)
EOT	(04)	DC3	(13)	EM	(19)	US	(1F)
ENQ	(05)	DC4	(14)	SUB	(1A)		

The TTY emulator also converts all 8-bit ASCII characters (above 7F in value) to 7-bit characters.

NOTE: You may change the characters filtered by the TTY emulator by modifying and using a translate file. See the preceding section, "Data Stream Filtering and Alteration," for complete details.

PASSTHRU

The BLAST PASSTHRU terminal is a "transparent terminal" that allows characters to be sent and received without any filtering, translation, or other special handling. PASSTHRU may be required to receive international characters or to operate a graphics terminal.

Special Considerations

There are some special considerations when using PASSTHRU:

◊ XON/XOFF flow control will still be honored.

◊ Setup functions normally available in Terminal mode are ignored; for instance, AutoLF IN and AutoLF OUT will not work.

◊ Local Echo will still work.

◊ BLAST will operate in either 7- or 8-bit mode.

◊ Hot Keys and *ATTN* Key sequences normally available in Terminal mode are ignored and will be sent as data (for a discussion of Hot Keys and a list of *ATTN* Key sequences, see "Keyboard Mapping Utility for 10.7x" on page 315 and "Attention Key Sequences" on page 346, respectively).

To interrupt Terminal mode and return to the BLAST menu system while in PASSTHRU, type:

ATTN ATTN (pause)

where "pause" indicates no keyboard input for a minimum of two seconds. This will allow the CTRL K sequence to be used in PASSTHRU Terminal mode.

Terminal Emulation with 10.7x

A "terminal" is a video monitor and keyboard that has been custom configured to generate and respond to formatting codes used by a particular computer system. For example, the VT100 terminal was originally designed to operate with Digital Equipment Corporation's VAX and PDP computers. Particular sequences of ASCII characters were defined to signal special actions, such as cursor movement, printer activation, and screen display behavior. In order to use your system as a terminal to a multi-user host like a VAX, your system must be able to produce and respond to the host's terminal control codes—a process called "terminal emulation." VT100 terminal emulation under BLAST Professional UNIX 10.7x allows your system to operate like a VT100 terminal.

The following emulators are available in BLAST Professional UNIX 10.7x:

BLAST Emulator	Terminal Emulated
PC ANSI	PC ANSI Color
VT52	DEC VT52
VT100	DEC VT100
VT220	DEC VT220
VT320	DEC VT320
ADM3A	Lear Siegler ADM3A
D80	Ampex Dialog 80
TV920	Televideo 920 series
WYSE50	WYSE 50+
WYSE60	WYSE 60

Most of these terminals feature unique keys to perform certain functions, such as the DO key on a VT220 terminal. Often it is possible to assign a standard key to perform the same task as a special terminal key. In other cases, it may be necessary to assign a combination of keys to perform the function—the DO key is mapped to CTRL F5, for example. Thus, your keystrokes are "mapped" or "routed" through BLAST's software to generate the required sequence of ASCII code for each terminal function. The default keyboard maps for all of BLAST's emulators are in Appendix B.

The default emulator for a session is specified in the Emulation setup field. To choose an emulator, use SPACEBAR to scroll through the available choices, then press ENTER to accept your choice. The VT, WYSE, and PC ANSI emulators have subwindows that appear automatically when you press ENTER. See Chapter 5 for more information on these setup subwindows.

PC ANSI Emulation

The BLAST ANSI emulator provides functional emulation of the IBM PC ANSI standard, including full color and extended attribute support. Choose PC ANSI for dialing Bulletin Board Systems or other computers that offer ANSI support.

DEC VT320, VT220, VT100, and VT52 Emulation

The BLAST VT emulators provide precise emulation of the DEC VT320, VT220, VT100, and VT52 terminals.

Supported Features

These emulators support the following features:

◊ All cursor positioning sequences and tab settings.

- All of the software-selectable operating states (or modes) available for the VT series of terminals, including standard ANSI and DEC private modes.

- The USASCII, UK, FRENCH, GERMAN, ITALIAN, SPANISH, and DANISH character sets. The default value is USASCII, which allows 8-bit data; the other character sets allow only 7-bit data.

- The DEC Supplemental Graphics and ISO Latin-1 character sets.

- Scrolling regions, line and character editing, and character attribute commands.

- All print operations, including Autoprint, Print Screen, and Printer controller (printer pass-through).

- Horizontal Scrolling control to accommodate 132-column display on a standard 80-column screen. The Scroll Left, Scroll Right, and Scroll mode keys may be used within Terminal mode and may be redefined with the blastkbd utility. To set the default mode for the number of columns to scroll, specify the column width in the VT Emulation setup subwindow.

Special Considerations

The following features are not supported by these emulators:

- Smooth scrolling.

- Downloadable character sets. These will be ignored by the application.

The following features are supported in the specified limited manner:

- Double-width characters are handled by displaying a single-width character and a space in a double-width position.

- Double-height characters are displayed in the top half of a double-height position.

- 132-column mode is supported by scrolling to the left or right on the screen as required to permit viewing of the entire 132 characters.

Refer to Appendix B for the Key Definition Chart.

WYSE 60/50, TV920, D80, and ADM3A Emulation

The WYSE 60/50, TV920, D80, and ADM3A emulators are the functional equivalent of the WYSE 60, WYSE 50+, Televideo family, Ampex D80, and the Lear Siegler ADM 3A terminals.

Supported Features

These emulators support the following terminal features:

◊ 80- or 132-column modes.

◊ Horizontal scrolling control using the cursor movement keys to accommodate a 132-column display on a standard 80-column screen. To set the default mode for the number of columns to scroll, specify the column width in the WYSE Emulation setup subwindow.

◊ Multiple pages (up to 4 pages; the default is 1).

◊ Split screens.

◊ Normal, dim, blink, blank, underline, and reverse attributes. (These attributes are embedded in the WYSE50+ emulation. They do not occupy a video position in the WYSE60 emulation.)

◊ Protected fields and display attributes of write-protected fields.

◊ Graphics characters.

◊ Print functions, including auxiliary and transparent modes.

◊ Editing functions.

◊ Block and character modes.

Special Considerations

These emulators do not support programming and displaying function-key labels and label lines.

Refer to Appendix B for the Key Definition Chart.

Transparent Print/Auxiliary Print

BLAST Professional UNIX 10.7x supports Transparent Print mode (data redirected to an attached printer as well as displayed on the screen) under the VT series, PC ANSI, WYSE 60/50, TV, D80, and

ADM3A emulations. In addition, the WYSE 60/50, TV, D80, ADM3A series supports Auxiliary Print mode (data redirected to an attached printer only).

BLAST Professional UNIX 10.7x recognizes the following codes for these functions:

WYSE 60/50, TV, D80, ADM3A

Transparent Print mode on:	ESC d #
Auxiliary Print mode on:	CTRL R
Transparent and Auxiliary Print mode off:	CTRL T

PC ANSI/VT

Transparent Print mode on:	ESC [5i
Transparent Print mode off:	ESC [4i

Keyboard Mapping Utility for 10.7x

Computer users sometimes encounter difficulties when emulating a terminal. For example:

- ◊ A key sequence meant to be passed to the remote computer is instead intercepted by an application (or the operating system).
- ◊ An emulator keymap is awkward for a particular application.
- ◊ Repetitive keystrokes are required for a particular application.
- ◊ A required key does not exist on the user's keyboard.

For BLAST Professional UNIX 10.7x, the keyboard mapping utility, blastkbd, helps address these problems. In blastkbd, there are three types of specially assigned key subsets in the BLAST key set: Soft Keys, BLAST Keys, and Hot Keys. In addition, blastkbd includes Emulator Maps and User-Defined Maps. Below is a brief description of each, followed by sections giving instructions on mapping and/or remapping each key set:

Soft Keys – Soft Keys allow you to send often-used character strings to a remote system with a single keystroke. The use of Soft Keys is described later in this chapter.

BLAST Keys – BLAST uses special key sequences to differentiate between local commands and characters meant for the remote system. The BLAST Keys perform local functions, such as exiting Terminal mode. The BLAST Keys are listed in Appendix B.

Hot Keys – Hot Keys access often-used functions from Terminal, Filetransfer, and Access modes. Hot Keys are essentially macros that activate BLAST menu commands and return you to your starting point with just a few keystrokes. Typing ALT F from a console in Terminal mode, for example, starts Filetransfer mode and automatically returns you to Terminal mode when file transfer is completed. The Hot Keys are listed in Appendix B.

Emulator Maps – These are the keyboard maps for the existing emulators within the BLAST program. With blastkbd, you can reroute existing functions to different keys on your keyboard. For a list of keys for the existing emulators, see “Terminal Emulation Keys for 10.7x” on page 348.

User-Defined Maps – You can create your own keyboard maps for different applications, keyboards, or users. Unlike the emulator maps, user-defined maps can specify functions as well as keys.

Running blastkbd

You can start blastkbd by typing blastkbd on the command line or, during a terminal session, by pressing *ATTN* M or *ATTN* E. *ATTN* M will display the main selection window (Figure 17-1) while *ATTN* E will take you directly to the specific map subwindow for the current emulator. From the emulator map, pressing ESC will return you to Terminal mode if that is where you started. If you started blastkbd from the command line, pressing ESC from the main blastkbd window will return you to the command line.

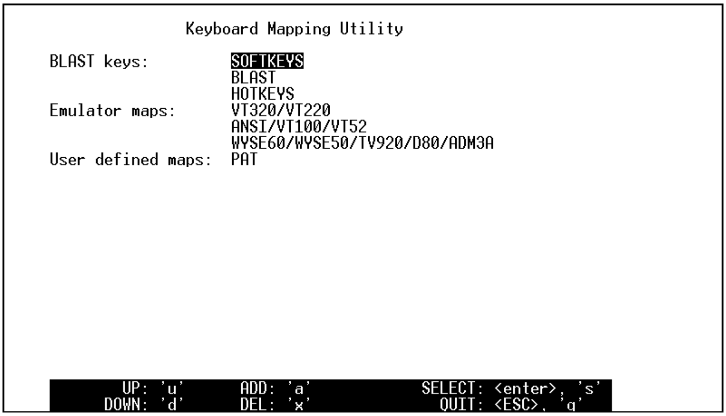


FIGURE 17-1

After you have edited a keyboard map or one of the BLAST special key sets, press S to save your changes. Press ESC if you wish to exit without saving the changes.

To select a BLAST special assigned key subset or a map from the blastkbd main window, use the commands described at the bottom of the window to highlight the desired selection and press ENTER.

NOTE: In the subwindows discussed below, some characters cannot be entered merely by pressing the corresponding key on the keyboard. The following table indicates how these characters are entered:

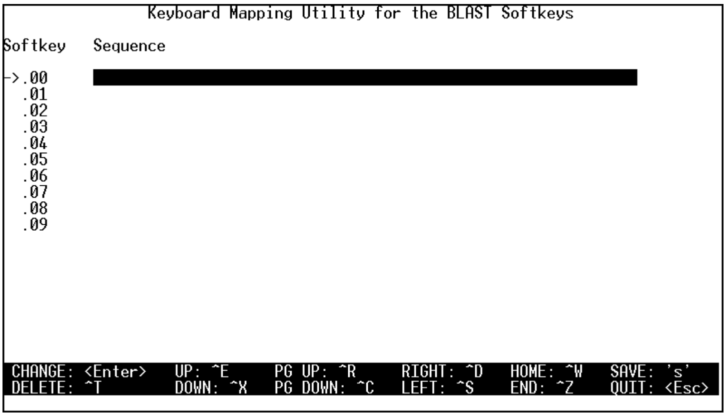
ESC	CTRL [
TAB	CTRL I
ENTER	CTRL M
^	^^

For example, to include the escape character in a key sequence, press CTRL [instead of pressing ESC. Some characters may appear in octal form, for example, CTRL^ may appear as \036.

Soft Keys

Many terminals offer a way of storing a set of often-used character strings that can be sent to the remote system with a single keystroke. BLAST provides this capability with Soft Keys. If you highlight Soft Keys in the main window and press ENTER, the Soft Key window (Figure 17-2) will appear:

FIGURE 17-2



To create a Soft Key, highlight the sequence for the Soft Key you have selected (0–9) and enter the text string to be sent to the remote system when that Soft Key is pressed. Each string can be up to 69 characters long.

BLAST allows ten Soft Keys. A Soft Key is activated from within Terminal mode with the following combination:

ATTN `Soft_Key_number`

where `Soft_Key_number` is the number key corresponding to the number of the text string. For example, `0` corresponds to .00 text string, `1` to the .01 text string, and `2` to the .02 text string.

BLAST Keys

You can also use blastkbd to modify the BLAST key subset. When you select BLAST from the BLAST Key set in the blastkbd main window and press ENTER, the BLAST Key subwindow (Figure 17-3) will appear.

FIGURE 17-3

```
                 Keyboard Mapping Utility for the BLAST Keys

  Function        Key 1                Key 2                Key 3
  --------        -----                -----                -----
->Cursor Up       <up>                 <ext><up>
  Cursor Down     <down>               <ext><down>
  Cursor Left     <left>               <ext><left>
  Cursor Right    <right>              <ext><right>
  Home            <home>               <ext><home>
  End             <end>                <ext><end>
  Page Up         <pgup>               <ext><pgup>
  Page Down       <pgdn>               <ext><pgdn>
  ESC key         <esc>
  Del Char        <del>                <ext><del>
  Insert Mode     <ins>                <ext><ins>
  Help            <f1>







 CHANGE: <Enter>   UP: ^E     PG UP: ^R     RIGHT: ^D    HOME: ^W    SAVE: 's'
 DELETE: ^T        DOWN: ^X   PG DOWN: ^C   LEFT: ^S     END: ^Z     QUIT: <Esc>
```

There are four columns—the first displays the functions supported by the BLAST Keys and the other three contain the key sequences you choose to perform that function. Up to three key sequences maybe specified for the same function. To remap a function, highlight one of the three key sequences to the right of the function and press ENTER. The message "Press any key to remap function..." is displayed. Type the key (or combination of keys) that will serve as this function. Repeat this process until you have remapped all the functions that you want; then press S to save your remappings and return to the blastkbd main window.

NOTE: BLAST keys do not change from emulator to emulator. For example, if you map the Cursor Down function as CTRL 2 in the

BLAST Keys subwindow while using the VT320 emulator, that sequence will also perform the same function if you switch to WYSE60 emulation.

You cannot use blastkbd to remap Attention (*ATTN*) Key sequences. The Attention Key can be remapped via the Attention Key setup field (page 73).

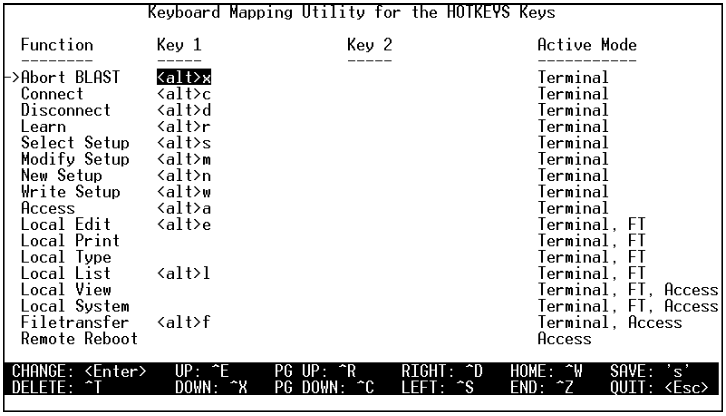
Hot Keys

If you select Hot Keys from the BLAST Key set in the blastkbd main window and press ENTER, the Hot Key subwindow will appear (Figure 17-4). Hot Keys override *all* other functions. For example, if you map both the VT Find key and the Filetransfer Hot Key to ALT F, pressing ALT F will *always* start Filetransfer mode and *never* act as the VT Find key.

To map or remap a function, highlight the first key sequence to the right of the function and press enter. The message “Press any key to remap function...” is displayed. Type the key (or combination of keys) that will serve as this function. Repeat this process until you have remapped all the functions that you want; then press S to save your remappings and return to the blastkbd main window.

NOTE: A Hot Key can only be mapped to a single keystroke. Any keystrokes entered into the second column will be ignored by BLAST.

FIGURE 17-4



Emulator Maps

Emulator maps act as links between your keyboard and the terminal you are emulating. For example, if you are using an AT extended keyboard through the VT320 emulator to a VAX minicomputer, the keymap will link the FI key to the VT320 PF1 function. To select an emulator, highlight the emulator in the blastkbd main window and press ENTER; the emulator subwindow will then appear. For example, if you select VT320/VT220, the subwindow in Figure 17-5 will appear.

FIGURE 17-5

```
              Keyboard Mapping Utility for the VT320/VT220 Keys

  Function       Key 1                Key 2                Key 3
  --------       -----                -----                -----
->Shift Tab      <shift><tab>
  Backspace      <ctrl><back>
  Del            <back>
  Cursor Up      <up>                 <ext><up>
  Cursor Down    <down>               <ext><down>
  Cursor Left    <left>               <ext><left>
  Cursor Right   <right>              <ext><right>
  Keypad 0       <keypad>0
  Keypad 1       <keypad>1
  Keypad 2       <keypad>2
  Keypad 3       <keypad>3
  Keypad 4       <keypad>4
  Keypad 5       <keypad>5
  Keypad 6       <keypad>6
  Keypad 7       <keypad>7
  Keypad 8       <keypad>8
  Keypad 9       <keypad>9

 CHANGE: <Enter>   UP: ^E     PG UP: ^R      RIGHT: ^D    HOME: ^W    SAVE: 's'
 DELETE: ^T        DOWN: ^X   PG DOWN: ^C    LEFT: ^S     END: ^Z     QUIT: <Esc>
```

To remap a function, highlight one of the three key sequences to the right of the function and press ENTER. The message “Press any key to remap function...” is displayed. Type the key (or combination of keys) that will serve as this function. Repeat this process until you have remapped all the functions that you want; then press S to save your remappings and return to the blastkbd main window. Up to three key sequences maybe specified for the same functio*n.*

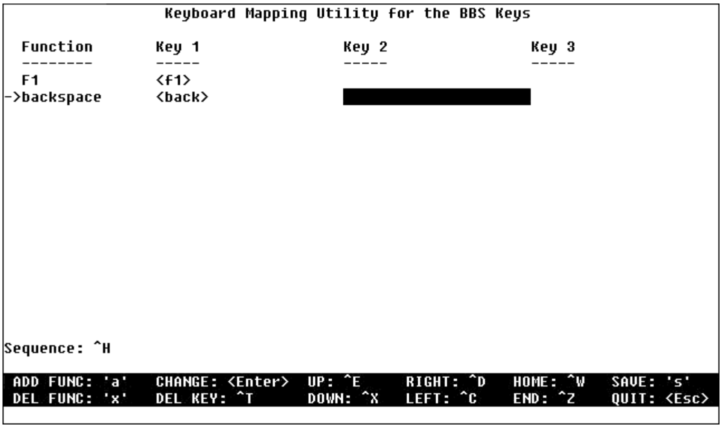
User-Defined Maps

A powerful feature of blastkbd is the option to create your own keyboard maps for different applications, keyboards, or users. For example, you can customize a map for a remote database application and save it under the name “data,” ready for use with BLAST. Once you have finished working with the database, you can load another map for another application.

To create a map, press A at the blastkbd main window. You will be prompted for the name of the new map. Pressing ENTER will add the new map name to the list of maps in the blastkbd main window (the map name will also appear as a selection in the Keyboard File setup field). Pressing ENTER again will display the mapping subwindow (Figure 17-6, next page). Unlike the emulator maps, user-defined maps allow you to specify the function as well as the keys.

The first step in assigning a function is to type the name of the function. If no functions have been assigned, simply type the name of the function in the field highlighted.To add a function, type A and then the name of the function you would like to add. After typing the name of the function, press ENTER. The first key sequence will automatically be highlighted. Type the key sequence for the function you have just added and press ENTER. At the bottom of your screen, you will be prompted for the ASCII control sequence. Type either the ASCII control sequence or octal value for that function (for a list, see Appendix D) and press ENTER. If you would like to add a second key sequence for the function or change an existing key sequence, highlight the desired key sequence to the right of the function and follow the same steps as you followed in entering the first key sequence. After you have finished mapping functions, press S to save your map.

FIGURE 17-6



Keyboard Map Selection in the Setup

All maps that you create are saved in a file called "blast.tdf." Each time that you start BLAST, it will search the current directory for blast.tdf. If it cannot be located, BLAST then checks BLASTDIR. You can easily assign separate keymaps for several users or applications by copying different blast.tdf files into each directory. When you run BLAST from within an application directory, the proper blast.tdf file will automatically be loaded.

To select a specific user map from within a given .tdf file, highlight the Keyboard File setup field and use the SPACEBAR to cycle through the map choices. If you would like a map to be loaded automatically on startup, save it as a part of the setup.

Chapter 18

Remote Control with 10.7x

What Is Remote Control?

For computers running BLAST Professional UNIX 10.7x *from the console,* the Remote Control features allow you to access and control the screen, keyboard, disk drives, and printer of a remote DOS PC running the BHOST program. Remote control is ideal for troubleshooting remote sites, training and supporting DOS operators, using DOS software—any time that you need complete control over a remote computer running DOS.

This chapter introduces basic concepts and guides you through the features of BLAST remote control. The *BHOST User Manual* describes how to set up the remote computer for control by BLAST. *Workstation users cannot remotely control a PC, but they can operate as limited DOS terminals and can transfer files through BHOST.*

Remote control allows a UNIX computer (the Controller) to completely control a DOS PC (the Host PC). The two systems can use a hardwire connection, a null modem cable, or modems communicating over a telephone circuit.

The Controller can run programs on the Host PC's hard drive, print documents, edit files, and more, as if the user were typing on the Host PC's keyboard. All video output and graphics are displayed simultaneously on both systems, with automatic translation between different video modes.

The Host PC

The Host PC runs the BHOST program, which operates transparently in the background. BHOST "watches" the communications port and, when a call comes in, prompts the caller for a user identification and password. Once the caller is logged in, BHOST makes the Host PC's services available to the Controller.

The Host PC has access to a number of security features, including login accounts, multiple control levels, call-back security, and a log file to record system activity.

Nearly all of the configuration for a remote control session takes place on the Host PC through SETBHOST, a special administration program that sets system defaults and keeps track of login accounts.

The Controller

The Controller runs BLAST and dials into a Host PC just as in an ordinary terminal session except that, once connected, the user of the Controller selects Access from the Online menu. Access mode allows complete control. (File Transfer Only and Terminal modes, discussed later, offer more limited control.)

In Access mode, the Controller can access a number of security features, including the ability to disable the Host keyboard, mouse, and screen during a session to prevent unauthorized operation.

Connecting to the Host PC

Connecting to the Host PC is the same as connecting to any other remote system. BLAST can automatically dial the phone and send your login ID and password to the Host PC. You may also perform this process manually.

Be sure that BHOST has been installed and configured on the Host PC before attempting to connect. See the *BHOST User Manual* for more information on installing and configuring BHOST.

Creating a BLAST Setup for BHOST

To automate your connection to a Host PC, create and save a new BLAST setup for your sessions with the Host PC (see Chapter 5 for a detailed description of setups). In the new setup:

◊ If you are using a modem, set the Phone Number to the phone number of the Host PC.

◊ Set the System Type field to `BHOST` if your BHOST account requires a login ID and password; set System Type to `PC` or `NONE` if your BHOST account does not require a login ID or password.

◊ If your BHOST account requires a login ID and password, enter these into the Userid and Password setup fields, respectively, exactly as they appear in SETBHOST on the Host PC. *These fields are case-sensitive.*

◊ Set Emulation to `TTY` or `VT320`.

◊ Set Protocol to `BLAST`.

◊ Set Packet Size to at least `200`, BHOST's minumum setting; the maximum setting is `4085`.

◊ In the BLAST protocol setup subwindow, set Compression Level according to the type of data you will transfer. Note that BHOST's compression level defaults to `1`. Any additional compression is determined by the amount of memory allocated by a `COMPBUF` assignment in BLAST.OPT on the Host PC. BHOST supports compression levels `0`–`4`.

Use the Write command from the Online menu to save the new setup.

Making the Connection and Logging On

Choose the new Host PC setup in your Setup Directory and select Connect from the Online menu. BLAST will make the connection, log onto the Host PC, and return to the Online menu.

NOTE: If your BHOST Account is set to `Dial Back`, BLAST will not return to the Online menu immediately. Instead, BHOST will disconnect after you log in and then dial your phone number from the Host PC. Once the connection has been re-established, BLAST will return to the Online menu.

Taking Control

How you take control of the Host PC depends on the Control mode setting in your BHOST Account. The possible settings are `Access`, `File Transfer Only`, and `Terminal`. The default Control mode is `Access`, which provides complete control over the Host PC.

Access mode – If your Control mode is set to `Access`, then press A from the Online menu to enter Access mode. You will then have complete control over the Host PC. All of your keystrokes are sent to the Host PC, and all of the Host PC's screen displays are sent to your system. Access mode offers a number of powerful features. See "Using Access Mode" below for complete details.

File Transfer Only mode – If your Control mode is set to `File Transfer Only`, then press F from the Online menu to enter BLAST Filetransfer mode. The BLAST Filetransfer menu will then appear, and you will be able to Send and Get files and execute operating system commands from the Local and Remote menus.

Terminal mode – If your Control mode is set to `Terminal`, then press T from the Online menu to enter Terminal mode. Terminal mode is limited to ASCII text display. Programs using graphics or full-screen text modes will execute, but the screen display will be corrupted and no error detection will be performed. Terminal mode requires special keyboard sequences to send control characters. See "Using Terminal Mode with 10.7x and 10.8x" on page 330.

Disconnecting from the Host PC

From Access mode – Press *ATTN* ESC to return to the Online menu.

From File Transfer Only mode – Press ESC to return to the Online menu.

From Terminal mode – Press ESC CTRL L to log off of the Host PC and then press *ATTN ATTN* to return to the Online menu. Select the Disconnect command to disconnect from the Host PC.

Using Access Mode

Access mode is very intuitive—all you have to do is type commands as if you were seated at the Host PC. Through the Access menu, BHOST provides several easy-to-use support features, such as:

◊ Split-screen Chat mode, for communicating interactively with the Host PC user.

◊ Two camera modes, one for taking “snapshots” of individual screens and one for recording “movies” of your session.

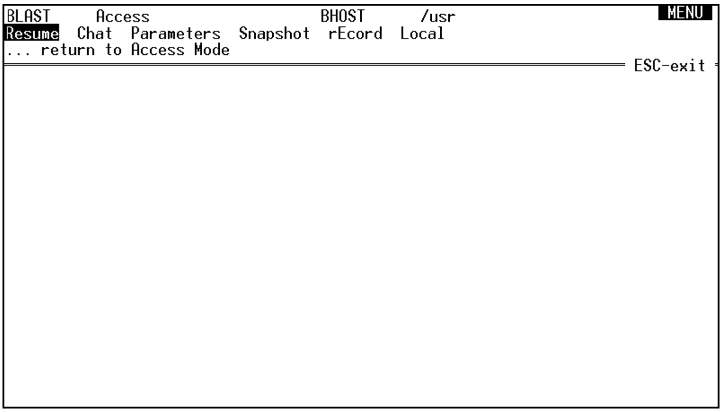
◊ A simple menu for fine-tuning your remote control settings.

◊ Hot Keys to start file transfers, exit to a local system shell, reboot the Host PC, and more.

The Access Menu

From Access mode, press *ATTN* to display the Access menu shown below in Figure 18-1.

FIGURE 18-1

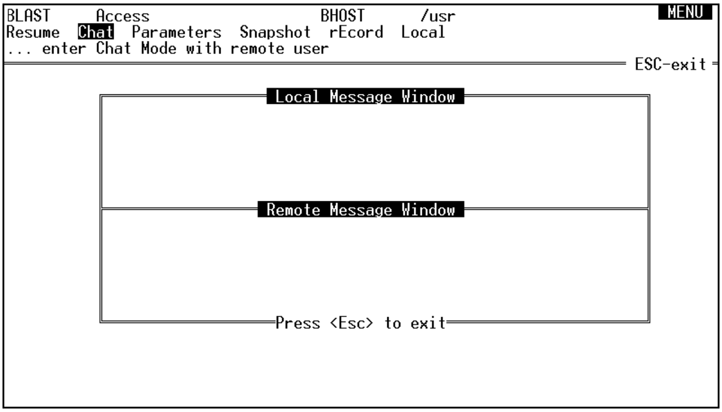


To select a command, press the capitalized letter in the command name or move the cursor over the command and press ENTER. Following is a description of each command:

Resume – Press R to return to Access mode.

Chat – Press C to start Chat mode. Chat mode allows the Host and Controller to type messages to each other on the Chat screen (Figure 18-2, next page), which is displayed on both the Controller and the Host screens. Either side may initiate a Chat unless the Host keyboard has been disabled. Once the Controller initiates a Chat, a disabled Host keyboard becomes active for the duration of the Chat.

FIGURE 18-2



The Chat screen contains two windows, one for the Controller's messages and one for the Host's messages. Both sides may type at the same time. Chat mode will terminate when either user presses ESC.

Parameters – Press P to display the Session Parameters window containing parameter fields that can be adjusted to improve BHOST performance (see "Modifying BHOST Settings with 10.7x" on page 332).

Snapshot – Press S to take a snapshot of the current screen. You will be prompted for a filename. After typing the filename and pressing ENTER, you will be returned to the Access menu. The current screen image will be saved to your current directory. If you type in a filename without an extension, BLAST automatically uses the extension ".001." Then, each time you take another snapshot, BLAST increments the extension by one (up to .099) and prompts you to save the new file.

BLAST saves text screens in standard ASCII file format and graphic screens in the .PCX format, which can be displayed with the View command from the Local menu or by a variety of third-party applications.

Record – Press E to record a "movie" of the screen appearance during your session (except Chat mode displays). You will be prompted for a filename. Type the filename and press ENTER. BLAST will then begin recording from the Host PC similar to VCR recording from a television. Escaping from the remote session screen for any reason will terminate the movie.

Movies can be replayed with the View command from the Local menu. By default, a movie is replayed at the same speed at which it was recorded. Press the up or down cursor keys during replay to speed up or slow down the movie. Note that movies can take up large amounts of disk space.

Local – Press L to display the Local menu for local system commands. This command is identical to the Local command available from the Offline and Online menus (see "The Local Menu" on page 58 for details).

Access Mode Hot Keys

The following subset of the regular BLAST Hot Keys are active during Access mode:

Function	Default Key Sequence
Chat mode	*
Local View	*
Local System	*
Remote Reboot	*
Snapshot	*
Parameters	*
Record a movie	*
Filetransfer mode	ALT F

***** To avoid potential conflicts with the programs running on the Host PC, these keys do not have default values. When you assign keys through the blastkbd utility, remember that the values you pick will not be available to the Host PC programs. For example, if you assign the ALT V key combination to the Local View function, then ALT V will never be sent to the Host PC, because it will be interpreted as a local command.

See "Hot Keys" on page 319 and Appendix B for more information on remapping Hot Keys.

Using File Transfer Only Mode with 10.7x and 10.8x

File Transfer Only mode allows more limited control of the remote PC than Access mode. This mode is available for both BLAST Professional UNIX 10.7x and 10.8x. In File Transfer Only mode, the local user can transfer files between the local computer and the PC using the Filetransfer menu and can perform remote commands using the Remote menu (page 60). To enter Filetransfer mode, press F from the Online menu.

Using Terminal Mode with 10.7x and 10.8x

If you are running 10.8x or are running 10.7x at an attached terminal, Terminal mode via BHOST allows you to act as a terminal to the Host PC. In Terminal mode, you will be able to run programs with line-mode ASCII text displays. Programs using graphics or full-screen text modes will execute, but the screen display will be corrupted and no error detection will be performed.

Starting and Ending Terminal Mode

If you are running 10.8x or are running 10.7x at an attached terminal, the Host PC Control mode should be set to `Terminal`. To begin Terminal mode, select Terminal from the Online menu. You can return to the Online menu at any time by pressing *ATTN ATTN*.

When you are ready to log out, *you must log out of Terminal mode correctly*: Press ESC CTRL L—you will automatically be logged out of BHOST on the Host PC. You can then return to the Online menu by pressing *ATTN ATTN*; then hang up the modem by selecting Disconnect.

Escape Sequences

Terminal mode requires special escape sequences to represent certain keys to the Host PC:

PC Key	Escape Sequence	
Left Arrow	ESC	F
Right Arrow	ESC	G
Up Arrow	ESC	T
Down Arrow	ESC	V
Home	ESC	H
End	ESC	E
Page Up	ESC	P
Page Down	ESC	Q
Insert	ESC	I
Delete	ESC	D
Numeric Keypad 5	ESC	. (period)
Numeric Keypad *	ESC	*
Break	ESC	S
Caps Lock	ESC	K
Num Lock	ESC	N

PC Key	Escape Sequence	
Numeric Keypad +	ESC	+
Numeric Keypad -	ESC	-
F1–F10	ESC 1	ESC 0
Esc	ESC	ESC
Esc	ESC	ESC
All keys released	ESC	SPACE
Ctrl	ESC	C
Alt	ESC	A
Left Shift	ESC	Z
Right Shift	ESC	/

The following escape sequences send special commands to BHOST:

PC Key	Escape Sequence
Filetransfer mode	ESC CTRL X
Repaint Screen	ESC CTRL R
Open Session Command window	ESC CTRL M
Log off	ESC CTRL L

Transferring Files to and from the Host PC

The BLAST protocol is available for transferring files to and from the Host PC. Your transfers will take place in the background on the Host PC, transparent to the Host PC user.

Starting Filetransfer Mode

There are several ways to initiate a file transfer to or from the Host PC. In each case, the BLAST Filetransfer menu appears, and you will be able to Send and Get files and execute operating system commands from the Local and Remote menus.

From the Online menu — Press F to start Filetransfer mode. Use this method if your BHOST account is set to `File Transfer Only`.

From Access mode — Press the ALT F Hot Key, or press *ATTN* ESC to return to the Online menu and then press F to start Filetransfer mode.

From Terminal mode — Press ESC CTRL X to start Filetransfer mode on the Host PC; then use one of the above methods to start Filetransfer locally.

Transferring Files

You may transfer files interactively (see “Performing Filetransfer Commands” on page 107) or via BLAST scripts (see “File Transfers with BLAST Session Protocol” on page 194).

Ending Filetransfer Mode

When you have finished transferring files, press ESC to end Filetransfer mode. If you started Filetransfer mode with a Hot Key, you will be returned to Access or Terminal mode. Otherwise, you will be returned to the Online menu.

Modifying BHOST Settings

Modifying BHOST Settings with 10.7x

If you are running 10.7x in Access mode, you may alter BHOST parameter settings by starting SETBHOST from the console. To start SETBHOST, type `SETBHOST` at the command line. For details on configuring BHOST via SETBHOST, see the *BHOST User Manual*. Note that the new settings will not take effect until BHOST has been restarted.

If you are in Access mode and have a `Superuser` account, you may alter BHOST session parameters by choosing Parameter from the Access Menu. The Session Parameters window (Figure 18-3) will then appear. Move through the fields by pressing the arrow keys; move through the options of a field by pressing SPACE or BACKSPACE.

FIGURE 18-3

```
BLAST     Access                  BHOST      /usr                        MENU
... <↑>-up <↓>-down <PgUp>-first <PgDn>-last
... press <Space>-next or <Backspace>-previous
                                                                     ESC-exit

    Session Parameters

             Scaling Ratio: 1:1
             Scan Interval: MEDIUM                Host Keyboard: ON
                 Sync Mode: ON                       Host Mouse: ON
      Special Kbd Handling: OFF                     Host Screen: ON
                                                   Host Printer: NONE
            Inactivity T/O: 120              Printer(s) Enabled: NONE
          Timeout Response: RESTART
         DCD Loss Response: RESTART
```

If you have a `User` account, you may change all of the settings on the Session Parameters screen except Inactivity T/O, Timeout Response, and DCD Loss Response. If you have a `Restricted` account, you will not be able to change any of the session parameter settings. If you change any of the settings via the Session Parameters window, the new settings will be in effect only for the duration of the session. Following is a description of the Session Parameter fields.

Scaling Ratio **[1:1]** 1:4 1:16 1:64

Specifies how the Host PC's graphics are scaled for screen updates. BHOST usually sends the entire Host screen to the Control PC. The Scaling Ratio allows certain portions of the screen to be omitted, resulting in much faster performance. Scaling Ratio only applies to graphics screens.

When Scaling Ratio is set to a value other than `1:1`, BHOST divides the Host PC screen into square grids and sends only the value of the first pixel in the grid. The Control PC then substitutes that value for each of the remaining pixels in the grid.

For example, when Scaling Ratio is set to `1:4,` BHOST sends only the first pixel of a 4-pixel grid. The Control PC writes that value for all four of the pixels in the grid.

`1:1` – the entire Host screen is sent to the Control PC.

`1:4` – the Host PC sends 1 pixel from a 4-pixel grid. (25% of the Host PC screen).

`1:16` – the Host PC sends 1 pixel from a 16-pixel grid. (6.25% of the Host PC screen).

`1:64` – the Host PC sends 1 pixel from a 64-pixel grid. (1.5% of the Host PC screen).

Use a higher Scaling Ratio `(1:4`, `1:16`, or `1:64`) when you want to see screens as quickly and image quality is not important.

Scan Interval NONE HIGH **[MEDIUM]** LOW

Specifies how often BHOST scans the Host PC's display to see if the display has changed since the last scan. If it has, BHOST rescans the display and sends the new screen to the Control PC.

The higher the Scan Interval, the more often the display is updated. A higher Scan Interval, however, usually means slower program speed since the foreground application on the Host PC must be interrupted for the scan, and each image must be sent to the Control PC.

`HIGH` – The Host screen is scanned 18.2 times per second (after each PC clock tick).

`MEDIUM` – The Host screen is scanned twice per second (after each 8 PC clock ticks).

`LOW` – The Host screen is scanned once per second (after each 18 PC clock ticks).

`NONE` – The Host screen is scanned only when the operating system is not updating the screen.

Sync Mode **[ON]** OFF

Specifies whether the Host PC and the Control PC screens will be synchronized.

When this field is set to `ON`, the Host PC screen is frozen while screen updates are sent to the Control PC. This mode completely synchronizes the two displays, but it slows the application speed.

When this field is set to `OFF`, the Host PC screen is not frozen, resulting in significantly faster performance. The Control PC, however, may miss some intermittent screen images.

Special KBD Handling ON **[OFF]**

Enables/disables Special Keyboard Handling.

IMPORTANT: This field should be set to `ON`.

Inactivity T/O 0 - 999 **[120]**

Specifies the number of seconds the Host PC will wait after no data has been sent or received before performing the action specified in the Timeout Response field (`RESTART` or `REBOOT`).

If this field is set to `0`, the Host PC will not time out.

If this field is set to `0` and the DCD Loss Response field is set to `IGNORE`, the Host PC modem may reset itself immediately after carrier is lost, even though BHOST is not ready to process incoming calls. In this case, BHOST will not restart without manual intervention, but the modem will continue to answer calls. To restart BHOST manually from the Control PC, first connect to the Host PC's modem; then enter Terminal mode and type:

```
;DISC.
```

Note that you will not be able to see your keystrokes. This sequence will interrupt the BLAST protocol and allow BHOST to restart—it may also cause the Host PC's modem to hang up. After BHOST has restarted, you may log on as usual.

Timeout Response **[RESTART]** REBOOT

Specifies the action that the Host PC will take if an Inactivity Timeout occurs. `RESTART` prepares the Host PC for the next caller, disconnecting the current user. `REBOOT` forces the Host PC to perform a warm boot just as if it had been physically rebooted with the CTRL ALT DEL sequence.

NOTE: If this field is set to `REBOOT`, the Host PC will not necessarily reload BHOST—you must specify BHOST in the Host PC's AUTOEXEC.BAT file to insure that the Host PC will be ready to answer incoming calls.

DCD Loss Response RESTART REBOOT **[IGNORE]**

Specifies the Host PC's actions if the modem's Data Carrier Detect (DCD) signal is lost during a session.

`RESTART` – restarts BHOST after DCD loss and prepares for the next caller. This is the recommended setting if you are using a modem and have an appropriate connection between the system and modem.

`REBOOT` – reboots the Host PC after DCD loss. Note that, with this setting, BHOST will not necessarily be reloaded. If BHOST is not loaded from the Host PC's AUTOEXEC.BAT file, the Host PC will remain at the DOS prompt when rebooted.

IGNORE – ignores DCD loss. In order for BHOST to detect DCD Loss through an external modem, the modem cable must support the DCD signal. All standard modem cables support this signal.

IMPORTANT: If DCD Loss Response is set to IGNORE and carrier is lost during a session, the Host PC modem may reset itself immediately, even though BHOST is not ready to process incoming calls. In this case, BHOST will not restart and the Host PC will not be able to process incoming calls until the Logon T/O or Inactivity T/O takes effect.

Host Keyboard **[ON]** OFF

Enables/disables the Host PC's keyboard. When this field is set to OFF, the Host Keyboard is completely disabled from the time BHOST is run; to regain control of the keyboard, you must reboot the Host PC or change this setting remotely. The Control PC may still initiate Chat Mode with the Host PC; in this case, the Host keyboard is enabled for the duration of the Chat.

IMPORTANT: If Host Keyboard is set to OFF and BHOST is started from the Host PC's AUTOEXEC.BAT, the Host PC's keyboard will remain disabled, even after rebooting. If this situation occurs, dial into the Host PC and change the Host Keyboard setting through SETBHOST.

This feature prevents unauthorized interference with a Control session.

Host Mouse **[ON]** OFF

Enables/disables the Host PC's mouse. When this field is set to OFF, the Host mouse is *completely disabled,* preventing unauthorized interference with a Control session.

Host Screen **[ON]** OFF

Enables/disables the Host PC's screen. When this field is set to OFF, the Host screen is completely disabled from the time BHOST is run, preventing anyone from seeing what is being sent to the Control PC's display.

When Host Screen is set to OFF, the Control PC may still initiate Chat Mode with the Host PC; in this case, the Host screen is enabled for the duration of the Chat.

IMPORTANT: If Host Screen is set to OFF and BHOST is started from the Host PC's AUTOEXEC.BAT, the Host PC's screen will remain disabled even after rebooting. If this situation occurs, try typing `BHOST /k` at the DOS prompt (you will not be able to see the characters on the screen). If that does not work, dial into the Host PC and change the Host Screen setting through SETBHOST.

This feature prevents unauthorized interference with a Control session.

Host Printer **[NONE]** LPT1 LPT2 LPT3

Specifies the Host PC printer to be used during a session. BHOST will monitor the printer port you specify here and redirect printing to the locations listed in the Printer(s) Enabled field.

If you plan to print during a session, set this field to the Host PC's printer port. You may notice a slight performance decrease.

If you do not plan to print during a session, set this field to `NONE`.

Printer(s) Enabled **[NONE]** CONTROL HOST BOTH

Specifies which printers will be active during a session. When an application issues a print command, the command will be executed on the printers specified here.

`NONE` – printing is disabled.

`CONTROL` – enables only the Control PC's default printer.

`HOST` – enables only the Host PC's printer as specified in the Host Printer field.

`BOTH` – enables both Host and Control printers.

Modifying BHOST Settings with 10.8x

If you are using a version of BLAST that does not support Access mode, such as 10.8x, you may alter the BHOST session parameters via the Session Command window (Figure 18-4, next page). To open the Session Command window, go to Terminal mode and press ESC CTRL M. Commands are entered as lines of text using the following format:

`parameter_command=value`

where *parameter_command* is one of the parameter commands listed in the table below and *value* is a valid setting for the parameter (see preceding section for setting options).

To check the current value of a session parameter, simply type the *parameter_command* for the parameter. For example, to display the current value for the Host Keyboard parameter, type:

```
keyboard
```

To see the values for all session parameters, type:

```
settings
```

Each parameter will be listed along with its current value. The following commands are available:

Parameter Command	Parameter
`DCDResp`	DCD Loss Response
`Inactimo`	Inactivity T/O
`Keyboard`	Host Keyboard
`Mouse`	Host Mouse
`Print`	Printer(s) Enabled
`PMouse`	Precision Mouse (unsupported; set to `OFF`)
`Printer`	Host Printer
`Screen`	Host Screen
`Scale`	Scaling Ratio
`Scan`	Scan Interval
`SKeyboard`	Special KBD Mode
`Sync`	Sync Mode
`TimoResp`	Timeout Response

To close the Session Command window, press ESC.

FIGURE 18-4

```
BLAST Host Session Command Mode
>settings

Current BLAST Host Settings

SCALE=1:1
SYNC=ON
SCAN=MEDIUM
PMOUSE=ON
SKEYBOARD=ON
INACTIMO=120
TIMORESP=RESTART
DCDRESP=RESTART
MOUSE=ON
SCREEN=ON
KEYBOARD=ON
PRINTER=NONE
PRINT=NONE
```

Appendix A

Error Messages

Introduction

The following is a list of BLAST error codes and a brief description of the cause of each error. Error messages for most versions of BLAST are included in this list. Even though they may not apply to the version running on the local computer, they may occur on the remote system.

BLAST Protocol Functions

20	loss of carrier during protocol logon
21	logon timeout (A BLAST protocol session was not established within the time specified by the BLAST Protocol Logon Timeout. See the Logon Timeout setup field description on page 84 for details.)
22	console interrupt the A*TTN* key was typed
23	inactivity timeout

	(A BLAST protocol session was terminated because of inactivity. See the Inactivity Timeout setup field description on page 85 for details.)
24	error in processing command file
25	cannot start BLAST on remote system
26	remote disconnect (The remote system timed out during a BLAST protocol session or the remote operator pressed the *ATTN* key.)
27	attempt to connect with an incompatible private network (There are special versions of BLAST that are limited to use within a particular network of systems. Use of these special versions outside of the network or use of a standard BLAST version within the network will give this message.)
29	connection control string timeout
30	loss of carrier during protocol connection

Transfer File Management

31	error-free file not found, or cannot be accessed (Often occurs because the file or directory does not have read permission.)
32	error-free file cannot be created (Often occurs because the file or directory does not have write permission.)
33	error-free file cannot be deleted (Check permissions on the directory.)
34	error occurred while closing the error-free file (This error occurs whenever BLAST cannot close an open file during Filetransfer mode.)
35	cannot position within the error-free file (This error occurs when BLAST cannot close an open file during Filetransfer mode.)
36	error occurred while reading the error-free file
37	error occurred while writing to the error-free file (Running out of disk space is a common cause of this error.)
38	size conflict
39	filename is too long or invalid
40	a file already exists with that name
41	error reading file directory (Check the permissions of the directory.)
42	error writing to disk; disk is full

48	permission denied (Your user profile on a multi-user system or the file attributes do not permit the current BLAST operation.)
49	transfer not allowed

Utility File Management

51	error opening a data file
52	error creating a data file
53	error deleting a data file
54	error closing a data file
55	error positioning within a data file
56	error reading from a data file
57	error writing to a data file
58	error in the size of a data file
59	error renaming a data file
60	directory specified in environment is invalid
61	SETUPDIR is not a directory
62	OPTDIR is not a directory

Scripting

65	script variable is READ-only
66	user-defined script error command
67	cannot find entry in modems.scr or systems.scr
68	no matching label for GOTO
70	error executing COMMAND.COM
71	all local commands complete
72	invalid file transfer switch specified
73	cannot overwrite or append
74	unknown file type
75	file already exists
76	too many open scripts
77	cannot load setup
78	setup already exists or cannot be created
79	not a valid directory
80	no setups found
81	no setup has been selected
82	upload cancelled
83	8-bit protocol requires an 8-bit channel; switching to 7-bit
84	packet size is too large; packet size too small for Access

85	remote control terminated by remote system
86	incompatible video mode
88	cannot initialize emulator
89	error printing, cannot open file

Command File Processing

90	error processing a command file (Syntax error in a BLAST script file while using video-suppress mode.)

Memory

105	error allocating memory

Initialization

100	error allocating memory from the BLAST memory pool
101	environment variable TERM is too large
102	cannot extract control strings from terminal information database (The TERM environment variable is not defined or the specified terminal type in TERM is incorrect.)
103	terminfo control string is too large
104	environment variable TERM is empty (Set the TERM environment variable. Depending on operating system you may have to “export” TERM.)
105	error allocating memory from the system
108	cannot load specified setup file (The setup file specified does not exist in either the current directory or the directory specified by the SETUPDIR environment variable.)
109	error in processing translate table update file
110	compression error
111	cannot execute a child process
112	error creating a pipe
113	cannot fork
117	cannot ioctl () the console port
118	cannot open the console port
119	cannot ioctl () the communications port

120	cannot open the communications port 1)You may have selected an invalid communications port. 2)Check the physical connection to the port. Make sure that the port specified is the actual port set up for communications. 3)The port may be in use or may not have been released by another system process. Reboot the computer and load only BLAST to test the physical connection. 4)The computer may be using an interrupt and/or base address that is not standard. Edit the BLAST.OPT to include proper address and IRQ. 5)The hardware flow control (RTS/CTS) or Carrier Detect signals may not be configured to handle the port signals directly. 6)Other applications may not have closed all ports when exiting. From the BLAST directory, type BLAST /I so that BLAST bypasses any checking of ports done by other applications.
121	a lock file exists for the communications port (Check the /usr/spool/uucp and/or /usr/spool/locks directories for a LCK.Portname file. Delete the lock file if appropriate. This is a System Administrator function.)
122	error in terminal definition
123	function not available in background mode
124	network error occurred
125	BLASTNMP.EXE not loaded
126	network drivers not loaded (If using TCP/IP, be sure that the name of the TCP/IP TSR matches the one specified in BLAST.OPT.)
127	Read error
128	unexpected signal
129–144	UNIX signal. Signal number is determined by subtracting 128 from the BLAST error number. This corresponds to UNIX signals 1–16.
150	Read error on comm port
151	Write error on comm port
210	compression error
253	internal error

Script Processor

300–399	syntax error in command

400	too many strings

Network

502	fatal network error; BHOST terminated

Appendix B

Key Definition Charts

BLAST Keys

BLAST menu functions are selected and controlled by the following keys.

Function	10.7x Console	10.7x Terminal	10.8x
Cursor Up	UP (↑)	CTRL E	CTRL E
Cursor Down	DOWN (↓)	CTRL X	CTRL X
Cursor Left	LEFT (←)	———	———
Cursor Right	RIGHT (→)	———	———
Move to First Field	PGUP	CTRL R	CTRL R
Move to Last Field	PGDN	CTRL C	CTRL C
Clear text in Field	CTRL T	CTRL T	CTRL T
CANCEL	ESC	ESC	ESC or CTRL A
HELP	F1	?	?
HELP (Terminal mode)	*ATTN* H	*ATTN* H	*ATTN* H

10.7x	BLAST menu functions can be remapped with the BLAST Keyboard Utility, blastkbd (see “Keyboard Mapping Utility for 10.7x” on page 315).

Attention Key Sequences

Attention Key sequences are only active from Terminal mode. For BLAST Professional UNIX 10.8x, terminal emulation must be set to TTY. The sequences cannot be remapped, but the Attention key can be redefined by entering a new setting in the Attention Key setup field (page 73).

ATTN *ATTN*	Return to the Online menu.
ATTN B	Send a break signal (also interrupts an active BLAST script).
ATTN C	Toggle Capture mode on or off.
ATTN H	Display Online Help

10.7x	*ATTN* E	Start blastkbd, the BLAST keyboard remapping utility, with the current emulator selected.
	ATTN M	Start blastkbd, the BLAST keyboard remapping utility.
	ATTN N	Reset XON/XOFF Pacing.
	ATTN P	Toggle printer logging on or off.
	ATTN 0–9	Start a BLAST Soft Key (digit is the Soft Key number).

Hot Keys

BLAST features Hot Keys for accessing certain functions from Terminal and Filetransfer modes (and Access mode in 10.7x). Not all functions are available from all modes (see chart below). Hot Keys are essentially macros that activate BLAST menu commands and return you to your starting point with just a few keystrokes. For example, in Terminal mode, typing ALT F in 10.7x and CTRL F in 10.8x starts Filetransfer mode and automatically returns you to Terminal mode when file transfer is completed.

Hot Keys are not available while BLAST scripts are running. To make Hot Keys active after an automated logon, be sure that the script command after `TERMINAL` is either `QUIT` or `RETURN`.

	Function	Key	Available Mode
10.7x	Abort BLAST	ALT X	Terminal
	Connect	ALT C	Terminal
	Disconnect	ALT D	Terminal
	Learn	ALT R	Terminal
	Select setup	ALT S	Terminal

Version	Function	Key	Available Mode
10.7x	Modify setup	ALT M	Terminal
	New setup	ALT N	Terminal
	Write setup	ALT W	Terminal
	Access	ALT A	Terminal
	Local Edit	ALT E	Terminal, FT
	Local Print	*	Terminal, FT
	Local Type	*	Terminal, FT
	Local List	ALT L	Terminal, FT
	Local View	*	Terminal, FT, Access
	Local System	*	Terminal, FT, Access
	Filetransfer	ALT F	Terminal, Access
	Remote Reboot	*	Access
	Snapshot	*	Access
	Chat	*	Access
	Parameters	*	Access
	Record	*	Access

FT=Filetransfer

* To avoid potential conflicts during remote control sessions, these keys do not have default values. When you assign keys through the BLAST keyboard mapping utility, blastkbd, remember that the values you pick will not be available to the Host PC programs during remote control sessions.

You can remap Hot Keys with blastkbd (see “Hot Keys” on page 319).

Version	Function	Key	Available Mode
10.8x	Filetransfer	CTRL F	Terminal
	Local System	CTRL N	Terminal
	Learn	CTRL R	Terminal

Terminal emulation must be set to TTY.

Terminal Emulation Keys for 10.7x

Press *ATTN* E or *ATTN* M from Terminal mode to view the blastkbd screen for these emulators (see “Terminal Emulation with 10.7x” on page 311).

DEC VT320 and VT220 Keys

Function	PC Key
Shift Tab	SHIFT TAB
Backspace	CTRL BACKSPACE
Del	BACKSPACE
Cursor Up	UP
Cursor Down	DOWN
Cursor Left	LEFT
Cursor Right	RIGHT
Keypad 0 – 9	keypad 0 – 9
Keypad -	keypad -
Keypad ,	keypad *
Keypad Enter Key	keypad +
Keypad .	keypad .
PF1 – PF4	F1 – F4
Hold Screen	F5
Print Screen	ALT P
Toggle Auto Print	CTRL PRTSC
Scroll Left	CTRL LEFT
Scroll Right	CTRL RIGHT
Scroll mode	(not mapped)
Find	INS
Ins Here	HOME
Remove	PGUP
Select	DEL
Prev Screen	END
Next Screen	PGDN
F6 – F12	F6 – F12
F13 – F20	CTRL F3 – CTRL F10
Help	CTRL F5
Do	CTRLF6
Shift 6 – F12	SHIFT F6 – F12
Shift F13 – F20	(not mapped)

DEC VT100 and VT52 Keys

Function	**PC Key**
Shift Tab	SHIFT TAB
Backspace	CTRL BACKSPACE
Del	BACKSPACE
Cursor Up	UP
Cursor Down	DOWN
Cursor Left	LEFT
Cursor Right	RIGHT
Keypad 0–9	keypad 0–9
Keypad -	keypad -
Keypad ,	keypad *
Keypad Enter Key	keypad +
Keypad .	keypad .
PF1–PF4	F1–F4
Hold Screen	F5
Print Screen	ALT P
Toggle Auto Print	CTRL PRTSC
Scroll Left	CTRL LEFT
Scroll Right	CTRL RIGHT
Scroll mode	(not mapped)

PC ANSI Keys

Function	**PC Key**
Backspace	BACKSPACE
Del	DEL
Cursor Up	UP
Cursor Down	DOWN
Cursor Left	LEFT
Cursor Right	RIGHT
PF1–PF4	F1–F4

WYSE 60, WYSE 50, TV920, D80, and ADM3A Keys

Function	**PC Key**
Backspace	CTRL BACK
Del	BACK
Enter	ENTER
Return	keypad +

Function	PC Key
Back Tab	SHIFT TAB
Print	ALT P
Send	ALT B
Scroll Lock	ALT Z
Unlock Kybd	ALT U
Cursor Up	ESC [A
Cursor Down	ESC [B
Cursor Left	ESC [D
Cursor Right	ESC [C
Home	HOME
Shift Home	CTRL HOME
Page Up	PGUP
Page Down	PGDN
Clear Line	END
Clear Screen	CTRL END
Del Char	DEL
Del Line	CTRL DEL
Ins Char	INS
Ins Line	CTRL INS
Ins/Replace	ALT I
F1–F12	F1–F12
F13–F16	CTRL F3–CTRL F6
Shift F1–Shift F12	SHIFT F1–F12
Shift F13–Shift F16	(not mapped)

Appendix C

Troubleshooting

1. **“I’m getting the message ‘A lock file exists for the comm port’ when I try to go online. What’s wrong?”**

   The communications port, or more specifically the serial port, is probably locked so that only one program at a time can use it. If a lock file exists for the serial port, BLAST assumes that another process is using it. Sometimes a lock file is left by a process that terminates abnormally. In this case, you must delete the lock file before BLAST can take control of the port. Lock files are kept in directories such as /etc, /var/spool/locks, /usr/spool/uucp, and /usr/spool/locks, depending on the particular version of UNIX that you have. Consult your System Administrator for the correct location of the lock file directory. Deleting lock files is normally a System Administrator function. See Chapter 2 for additional information.

2. **“I get the message ‘Unable to open communications port’ when I try to go online. What’s wrong?”**

   Make sure that you have specified the port correctly in your setup and that another process is not currently using the port. For example, a getty might be running on the communications port or a lock file might exist even though the process that created the lock file has terminated. See Chapter 2 for additional information.

3. **"Soon after going online I get the message 'Read error' or 'read error on port' and BLAST is terminated. What happened?"**

   The most common reason for this error is that another process in your system has tried to access the serial port while BLAST was using it. Check to be sure that programs like cu, uucp, getty, or ttymon are not attempting to run simultaneously on the same port as BLAST. Make sure that these programs are set up to recognize lock files and share system resources smoothly. For more information on ttymon, refer to Chapter 2 of this manual. Occasionally, this error can occur if BLAST is unable to communicate with the tty from which BLAST is being run.

4. **"Can someone log onto my system and exchange files with my system without my help?"**

   The user must have a legitimate ID and password on your system and the environment variables must be properly set to allow the user to execute BLAST. For more information on login, see Chapter 2. After logging in, the remote user should invoke BLAST in host mode, with the -h switch:

   blast -h

   When the message

   ;starting BLAST protocol

   appears, the user should initiate BLAST Filetransfer mode on the remote system (see Chapter 6). This feature is available in BLAST protocol only. Several of the other supported protocols can operate in a much more limited pseudohost mode (see "BLAST Operation as a Pseudohost With 10.8x" on page 204).

5. **"I can connect and log in normally, but when I enter BLAST Filetransfer mode, my files won't transfer. Why is this happening?"**

   You may have either a mismatched filetransfer channel or a flow control problem. Both sides of the connection must use the identical channel width. The 7-Bit Channel setup field must have the same setting on both sides of the connection. Flow control must be established correctly between each computer and its modem and between modems (see "Flow Control" on page 30).

6. **"What's a quick way to get started with scripting?"**

   Use BLAST's Learn mode (page 176) to build a script as you go through the steps of a process interactively.

7. **"After making a connection, the line goes dead. I can tell that the modems are still connected, but no data is being transmitted."**

   Make sure that both sides of the connection are using the same communications parameters, such as parity, data/stop bits, and flow control. If you cannot see anything that you type on your screen but your data is being transmitted correctly, change the Local Echo setting to YES.

8. **"Is there a way to send my own initialization string to the modem?"**

   You can communicate directly with the modem while in Terminal mode, or you can write your own script (see "Sample Modem Script" on page 214).

9. **"What are typical modem settings required by BLAST?"**

   DTR Normal
   CD Normal
   Verbal Result Codes
   Display Result Codes
   Modem Echoes Commands
   Enable AT Command Set

Appendix D

The ASCII Character Set

D	H	O	M	D	H	O	M	D	H	O	M	D	H	O	M
0	00	00	nul	32	20	40	space	64	40	100	@	96	60	140	‘
1	01	01	soh	33	21	41	!	65	41	101	A	97	61	141	a
2	02	02	stx	34	22	42	”	66	42	102	B	98	62	142	b
3	03	03	etx	35	23	43	#	67	43	103	C	99	63	143	c
4	04	04	eot	36	24	44	$	68	44	104	D	100	64	144	d
5	05	05	enq	37	25	45	%	69	45	105	E	101	65	145	e
6	06	06	ack	38	26	46	&	70	46	106	F	102	66	146	f
7	07	07	bel	39	27	47	’	71	47	107	G	103	67	147	g
8	08	10	bs	40	28	50	(	72	48	110	H	104	68	150	h
9	09	11	ht	41	29	51	)	73	49	111	I	105	69	151	i
10	0A	12	lf	42	2A	52	*	74	4A	112	J	106	6A	152	j
11	0B	13	vt	43	2B	53	+	75	4B	113	K	107	6B	153	k
12	0C	14	ff	44	2C	54	,	76	4C	114	L	108	6C	154	l
13	0D	15	cr	45	2D	55	-	77	4D	115	M	109	6D	155	m
14	0E	16	so	46	2E	56	.	78	4E	116	N	110	6E	156	n
15	0F	17	si	47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20	dle	48	30	60	0	80	50	120	P	112	70	160	p
17	11	21	dc1	49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22	dc2	50	32	62	2	82	52	122	R	114	72	162	r
19	13	23	dc3	51	33	63	3	83	53	123	S	115	73	163	s
20	14	24	dc4	52	34	64	4	84	54	124	T	116	74	164	t
21	15	25	nak	53	35	65	5	85	55	125	U	117	75	165	u
22	16	26	syn	54	36	66	6	86	56	126	V	118	76	166	v
23	17	27	etb	55	37	67	7	87	57	127	W	119	77	167	w
24	18	30	can	56	38	70	8	88	58	130	X	120	78	170	x
25	19	31	em	57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32	sub	58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33	esc	59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34	fs	60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35	gs	61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36	rs	62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37	us	63	3F	77	?	95	5F	137	-	127	7F	177	del

D – decimal; H – hexadecimal; O – octal; M – mnemonic

The chart below is a list of the standard ASCII control codes—with the decimal, hexadecimal, and octal values; the ASCII mnemonic; the key sequence, and a short explanation.

D	H	O	M	Sequence	Explanation
0	00	00	nul	<ctrl> @	used for padding
1	01	01	soh	<ctrl> A	start of header
2	02	02	stx	<ctrl> B	start of text
3	03	03	etx	<ctrl> C	end of text
4	04	04	eot	<ctrl> D	end of transmission
5	05	05	enq	<ctrl> E	enquire
6	06	06	ack	<ctrl> F	positive acknowledgement
7	07	07	bel	<ctrl> G	audible alarm
8	08	10	bs	<ctrl> H	backspace
9	09	11	ht	<ctrl> I	horizontal tab
10	0A	12	1f	<ctrl> J	line feed
11	0B	13	vt	<ctrl> K	vertical tab
12	0C	14	ff	<ctrl> L	form feed
13	0D	15	cr	<ctrl> M	carriage return
14	0E	16	so	<ctrl> N	shift out
15	0F	17	si	<ctrl> O	shift in
16	10	20	dle	<ctrl> P	data link escape
17	11	21	dcl	<ctrl> Q	device control 1 (resume output)
18	12	22	dc2	<ctrl> R	device control 2
19	13	23	dc3	<ctrl> S	device control 3 (pause output)
20	14	24	dc4	<ctrl> T	device control 4
21	15	25	nak	<ctrl> U	negative acknowledgement
22	16	26	syn	<ctrl> V	synchronization character
23	17	27	etb	<ctrl> W	end of text block
24	18	30	can	<ctrl> X	cancel
25	19	31	em	<ctrl> Y	end of medium
26	1A	32	sub	<ctrl> Z	substitute
27	1B	33	esc	<ctrl> [	escape
28	1C	34	fs	<ctrl> \	frame separator
29	1D	35	gs	<ctrl>]	group separator
30	1E	36	rs	<ctrl> ^	record separator
31	1F	37	us	<ctrl> _	unit separator

D – decimal; H – hexadecimal; O – octal; M – mnemonic

Appendix E

Autopoll

The Autopoll Script

BLAST features Autopoll, a sample script that allows your unattended system to call a series of remote computers and exchange information. Autopoll performs the following tasks:

◊ reads a list of sites to be polled,

◊ connects to each site,

◊ executes a transfer command file to transfer files,

◊ disconnects,

◊ scans the log file to determine which transfers were successful,

◊ builds retry files as required,

◊ and adds the results to a status file.

Autopoll checks carefully for errors while polling. If an error is found, the problem site is scheduled to be retried. Only the file transfer commands that failed are attempted again.

Installing Autopoll

Autopoll consists of eight scripts that were copied into your BLAST directory when the BLAST program was installed on your system. The scripts are:

autopoll.scr – master script.
autoinit.scr – initializes variables and files.
autoierr.scr – reports initialization errors.
autodisp.scr – draws screen displays.
autoline.scr – reads site information.
autopsnd.scr – checks log for status of `SEND`s.
autoprcv.scr – checks log for status of `GET`s.
autoparx.scr – updates status files.
autosw.scr – strips file transfer switches off `@filename`. (10.8x only)

The scripts may be moved to any convenient directory in your system. For instance, you could segregate Autopoll from other BLAST files by creating a poll directory:

```
cd /usr/blast
mkdir poll
mv auto*scr poll
```

In addition to these script files, you must have a BLAST setup called "autopoll" located in the BLAST Setup Directory. It must include a valid communications port or hunt file and other connection information such as modem type and baud rate. You may also specify the script autopoll.scr in the Script File field of the setup, simplifying the command line to start Autopoll.

Starting Autopoll

Autopoll must be started from the directory in which the Autopoll scripts and support files (site and transfer command files) are found. If blast.exe is not in this directory, you need to add the full path for blast.exe to your PATH (see "Setting PATH, BLASTDIR, and SETUPDIR" on page 8) or give the full path in the command line. If autopoll.scr has been entered in the Script File field of the autopoll setup, the format for invoking Autopoll from the command line is:

blast autopoll *max_cycles site_file* [*start_time*]

If autopoll.scr has not been entered in the Script File field of the setup, the command line must explicitly include the script:

blast autopoll -sautopoll.scr *max_cycles site_file* [*start_time*]

Other command line switches may be required under certain conditions. For example, if you intend to run Autopoll from cron, you must disable terminal output by including the -b or -n switch on the command line (see "Autopoll under cron" on page 368).

The command line parameters have the following meaning:

autopoll	the autopoll setup.
-sautopoll.scr	the autopoll script.
max_cycles	the maximum number of attempts to complete all specified transfers.
site_file	the filename "stub" (the part of the filename before the extension) of the site description file.
[*start_time*]	[optional] the time, in 24-hour format, that Autopoll will begin polling. The WAIT UNTIL command in BLASTscript requires the 24-hour format. If this parameter is omitted, Autopoll begins polling immediately.
[TRACE]	[optional] the command to enable a capture file of the entire polling session. The capture file contains the text of login dialogs, modem initialization commands, and so forth. This feature is used primarily for troubleshooting.

Here are some example command lines:

```
blast autopoll 3 retail 10:45
blast autopoll 1 northwest -n &
blast autopoll 2 daily 1:05 TRACE
```

In the first example, a maximum of three attempts will be made to poll the sites listed in the site file retail.dat starting at 10:45 am. Notice that the command line specifies just the stub "retail" of the site filename retail.dat. (Autopoll appends a variety of extensions to the filename stub to specify the names of special files.)

In the second example, one attempt will be made immediately to poll the sites in northwest.dat, BLAST will suppress its terminal output (-n), and UNIX will place the BLAST job in the background (&).

In the third example, a maximum of two attempts will be made to poll the sites listed in the site file daily.dat starting at 1:05 am, and a trace of the polling session will be made.

NOTE: Versions of BLAST before 10.7.5 do not support the `@SETUPDIR` reserved variable. If you are running an earlier BLAST version, you must include a reference to SETUPDIR on the command line:

blast autopoll -sautopoll ${SETUPDIR}/ *max_cycles site_file* [*start_time*]

The Site File

The site file is the "master list" of information about the sites to be polled. Site files may use any valid filename, but the extension must be .dat. Each line in the site file holds the parameters needed to connect to and transfer files to and from one site. Each line, or site record, consists of five fields separated by exclamation marks, also called "bangs," in the form:

`setup_name``!`*`site_name`*`!`*`phone_number`*`!`*`baud_rate`*`!`*`TCF_name`*

where

`setup_name` specifies a setup to be used for polling. If omitted, the field defaults to autopoll.

`site_name` contains a descriptive label for the site. If omitted, the field defaults to the Description field of *`setup_name`*.

`phone_number` specifies the phone number to be used for the site. If omitted, Autopoll uses the Phone Number field of *`setup_name`*.

baud_rate specifies the baud rate to be used for this site. If omitted, Autopoll uses the Baud Rate field of *setup_name*.

TCF_name specifies the transfer command file (TCF) to be used for this site. If omitted, this field defaults to autopoll.tcf.

Each line must contain four bangs. Any fields that are to be skipped must be indicated by consecutive bangs (!!). Blank lines and lines beginning with a space, tab, or pound sign (#) will be skipped, so you may freely comment your site file using these characters. Lines may not exceed 100 characters in length. Some example record lines are as follows:

[the ruler is shown to indicate column position]

```
1        10        20        30        40        50
|...|....|....|....|....|....|....|....|....|....|
!Blaster!1(919)542-0939!!
store06!!!!nightly.tcf
NewYork!Albany!782-8311!19.2!ny.tcf
```

In the first site record, no setup is specified, so autopoll.su will be loaded. The site name will be “Blaster,” overriding the Description field of the setup. The phone number will be 1(919)542-0939. The baud rate will be taken from the setup because that field is blank, and the transfer command file will default to autopoll.tcf.

In the second record, the setup store06.su will be loaded. The site name, phone number, and baud rate will default to the values given in store06.su. The transfer command file will be nightly.tcf.

In the last record, the file NewYork.su will be loaded. The site name will be “Albany,” the phone number will be 782-8311, the baud rate will be set to 19.2 kbps, and the transfer command file will be ny.tcf.

Transfer Command File

Autopoll uses a standard transfer command file (TCF) to specify files to be sent and received. You may use a unique TCF for each site listed in your site file, or you may use one TCF for multiple sites. For a complete description of the Transfer Command File, see “Transfer Command File” on page 115.

IMPORTANT: Autopoll treats wildcards and remote commands (such as remote print and remote rename) as "try once" specifications. These transfers and commands are attempted during the first cycle only. Even if errors occur, Autopoll does not attempt the transfers or commands again. For this reason, wildcards and remote commands should be used with caution.

Overview of Autopoll Script Actions

A brief overview of the basic actions of the autopoll scripts follows to give users a clearer understanding of the Autopoll process. Much of the error checking, which comprises most of the scripts, is not included.

1. Autopoll.scr starts, reads the command line parameters, and puts them into variables.

2. If an error is found, autopoll.scr calls autoierr.scr, which reports errors and terminates the Autopoll session.

3. If no errors are found, autopoll.scr calls autoinit.scr, which initializes variables and files. Specifically, using the stub of the site file, autoinit.scr sets variables that allow Autopoll to create retry and summary files and to find stop and banner files (see "Other Files Using the Filename Stub" on page 366) to be used in the Autopoll session. Autoinit.scr then returns control to autopoll.scr.

4. Autopoll.scr calls autoline.scr, which reads and interprets the site file line by line for `@SYSDESC`, `@PHONENO`, `@WORKTCF`, and `@LOGFILE` and returns control to autopoll.scr.

5. Autopoll.scr calls autodisp.scr, which then displays on-screen status information during polling and then returns control to autopoll.scr.

6. Autopoll.scr uses variables gleaned from the site file by autoline.scr to begin file transfer of the first site. After it finishes the first filetransfer session, autopoll.scr loops back to call autoline.scr to get information for the next filetransfer session until it finishes attempting the complete cycle of file transfers.

7. Autopoll.scr calls autoprcv.scr (which calls autosw.scr) and autosnd.scr to check the error-free log file for errors generated in the filetransfer sessions.

8. Autopoll.scr calls autoparx.scr (which calls autosw.scr) to update the screen and status file.

9. If more than one cycle is designated in the command line, autopoll.scr uses the updated status file to retry any files that failed in the first cycle.

10. Steps 7–9 are repeated until all files have been successfully transferred or until the number of cycles designated in the command line has been completed.

11. Autopoll.scr quits

NOTE: Autopoll.scr also calls any userscripts that may be created. See "User-Supplied Scripts" on page 370 for details on creating these scripts and on the points at which autopoll.scr calls these scripts.

Configuration Example

Assume that you have been asked to set up a polling network for a client who has a central UNIX system and two remote UNIX sites. How do you set up Autopoll for this configuration? First, you install BLAST on the central and remote sites and verify that connections can be made reliably. This step is best performed interactively, that is, while you are at the central system issuing commands directly to BLAST. When you are satisfied that BLAST is correctly installed, you need to create the following:

◊ setup files

◊ the site file

◊ transfer command files

The Setup Files

Suppose the sites are configured as follows:

Site name	Phone	Login, password
Sam's Discount Mart	542-0307	buz, apollo11
Metro Army Surplus	542-5694	neil, saturn5

Because the logins are different, different BLAST setup files are needed for each site. The setups, called "sam" and "metro," are created by running BLAST at the central site (see “Creating a New Setup” on page 64).

The Site File

Using the setups, you could write a site file named retail.dat:

```
1        10         20         30         40         50
|...|....|....|....|....|....|....|....|....|....|
 Retail Site List for My Polling Network
sam!Sam's Discount!542-0307!!sams.tcf
metro!Metro Army Surplus!542-5694!!metro.tcf
```

The first line of the file is treated as a comment because it begins with a space. The last two lines are the actual site records. In this case, the site records may be duplicating information already specified in the Phone Number and Description fields of the setups. If so, the site records could be simplified:

```
1        10         20         30         40         50
|...|....|....|....|....|....|....|....|....|....|
 Retail Site List for My Polling Network
 (Phone number and Description loaded from setups)

sam!!!!sams.tcf
metro!!!!metro.tcf
```

The site file now has an additional comment line (five lines altogether); otherwise it is equivalent to the previous site list.

Transfer Command Files

According to the site list, a transfer command file called sams.tcf will be executed when Autopoll connects to Sam’s Discount Mart, and the transfer command file metro.tcf will be executed when Autopoll connects to Metro Army Surplus.

Suppose you need to get two files from Sam and send one to him. The file sams.tcf might look like this:

```
1        10         20         30         40         50
|...|....|....|....|....|....|....|....|....|....|
+/usr/buz/acq12.txt /usr/client/sam1
+/usr/buz/wk_82 /usr/client/sam2
/usr/tmp/message /usr/tmp/read_me/OVW
```

As explained in “Transfer Command File” on page 115, the “+” sign in column 1 of a line signifies that BLAST will perform a `GET`. Thus, in the file sams.tcf above, BLAST will get /usr/buz/acq12.txt and give it the local filename usr/client/sam1. BLAST will also get /usr/buz/wk_82 and give it the local filename /usr/client/sam2. The absence of a “+” in the last line of the TCF signifies that BLAST will perform a `SEND`. Thus, BLAST will send usr/tmp/message and give it the filename /usr/tmp/read_me on the remote system. The added /OVW switch signifies that BLAST will overwrite an existing file of the same name on the remote system (see “File Transfer Switches” on page 111 for more information about filetransfer switches).

Metro.tcf is similar to sams.tcf:

```
1        10        20        30        40        50
|...|....|....|....|....|....|....|....|....|....|
+/usr/neil/acq12.txt /usr/client/metro1
+/usr/neil/wk_82 /usr/client/metro2
/usr/tmp/message /usr/tmp/read_me/OVW
```

Where to Save Autopoll Files

The site file retail.dat and transfer command files sams.tcf and metro.tcf are created using a standard text editor and *saved as text files only* in the same directory as the Autopoll scripts.

IMPORTANT: Autopoll script files, transfer command files, and site files must be stored in the same directory, which must be your current working directory.

.

Starting Autopoll

With the required files ready, the BLAST command line to start Autopoll might be:

```
blast autopoll 3 retail
```

which specifies a maximum of three attempts to complete the polling session with retail.dat.

Other Files Using the Filename Stub

Autopoll distinguishes several special files by appending different extensions to the site filename stub. The extensions for retail.dat are listed below:

Extension	Created by	Meaning	Example
.dat	user (required)	Site file	retail.dat
.stp	user (optional)	Stop file	retail.stp
.hdr	user (optional)	Banner file	retail.hdr
.log	Autopoll	Short summary file	retail.log
.prn	Autopoll	Long summary file	retail.prn

Site File

The site file (retail.dat) is the master list of information about the sites to be polled.

Stop File

The stop file (retail.stp) is an optional file the user can create that allows BLAST to exit prematurely but gracefully from a polling session. Autopoll checks for the existence of the stop file in the Autopoll directory before each connection to a site. If the file is found, the polling session is terminated.

For example, suppose you want to halt Autopoll because you have found out that the files to be transferred to the last 10 sites of a polling session have been corrupted as a result of an error in database reporting. Creating a stop file—a file with the stub of the site file and the extension ".stp"—will allow BLAST to quit the polling session gracefully instead of connecting to the last 10 sites.

Since the existence of the stop file—and not its contents—signify to BLAST that a session should be terminated, the contents of the file are irrelevant. A convenient way to create a stop file is with the UNIX touch command:

```
touch retail.stp
```

To ensure the completion of future transfers for the site file, Autopoll deletes the stop file before exiting.

Banner File

The banner file (retail.hdr) is an optional file created by the user. Autopoll prints the banner file prior to printing the summary file at the end of polling. Printing is performed by the BLASTscript `LPRINT` command. You might want this file to contain special text or graphics to distinguish the summary file within a large queue of printouts.

Long and Short Summary Files

Autopoll maintains two summary files, a long summary file and a short summary file. Prepared by Autopoll but not printed, the long summary file (retail.prn) is helpful for troubleshooting. Printed automatically at the end of polling, the short summary file (retail.log) is most helpful when polling goes well because a quick glance will confirm a successful polling session. The files are saved in the Autopoll directory.

A typical short summary file looks like this:

```
*********************  02/09/96  11:15:29  *********************
Cycle 1
  1. FAILED:Sam's Discount < Error transferring 3 file(s). >
  2. success: Metro Army Surplus

Cycle 2
    1.  success: Sam's Discount
****************************************************************
Note: check retail.prn for complete session information.
```

A typical long summary file looks like this:

```
02/09/96  **************************************************************
11:15:33  * Cycle: 1    Site: 1
          *
          *  Name:  Sam's Discount
          * Phone:  542-0307
          *   TCF:  sams.tcf
          *   Log:  C1S001.log
          *
          *————————— SESSION INFORMATION —————————
          * Filetransfer error -8: DCD lost during transfer
          * Error transferring 3 file(s). Log file follows:
          *
          * ****  BLAST Professional UNIX 10.7.3 on remote system [uov]
          * LOSS OF CARRIER, ending Filetransfer
          * File transfer interrupted, 12% of file acq12.txt received
          **************************************************************

02/09/96  **************************************************************
11:16:30  * Cycle: 1    Site: 2
          *
          *  Name:  Metro Army Surplus
          * Phone:  542-5694
```

```
        *   TCF:  metro.tcf
        *   Log:  C1S002.log
        *
        *———————————— SESSION INFORMATION ——————————
        * No errors encountered.
        * Log file has been deleted.
        **************************************************************

02/09/96 **************************************************************
11:18:49 * Cycle: 2    Site: 1
        *
        *  Name:  Sam's Discount
        * Phone:  542-0307
        *   TCF:  C1S001.tcf
        *   Log:  C2S001.log
        *
        *———————————— SESSION INFORMATION ——————————
        * No errors encountered.
        * Log file has been deleted.
        **************************************************************

02/09/96 **************************************************************
11:20:41 * Polling complete: all sites polled successfully.
        **************************************************************
```

Autopoll under cron

cron is a UNIX scheduler. To run Autopoll under cron, you will need to create a crontab that launches BLAST. The crontab usually specifies a shell procedure that starts BLAST after setting some environment variables like BLASTDIR and SETUPDIR.

A typical crontab could consist of a single entry:

```
10 2 * * * /usr/doug/gopoll
```

This line instructs cron to start the program /usr/doug/gopoll at 2:10 every day. The gopoll program may be a shell procedure, such as:

```
PATH=$PATH:/usr/blast
BLASTDIR=/usr/blast
SETUPDIR=/usr/doug
BPRINTER="lp -d laser %s"
export PATH BLASTDIR SETUPDIR BPRINTER
cd /usr/blast/poll
blast autopoll 6 nightly -b
```

This shell procedure sets environment variables needed by BLAST and ensures that the correct directory has been entered before starting BLAST. Of course, on your system these environment variables probably have different values. The -b switch disables terminal output. Consult the cron man page of your UNIX system for more information about cron and crontab.

Tips and Tricks

Following are a few tips and tricks to help insure successful execution of Autopoll:

Keep it Simple

Polling sessions can quickly become complicated if several file transfers must be performed over a large network of remote sites. Create simple but sensible directory structures to support the polling network. As a rule of thumb, command files should contain lines no longer than 80 characters so that they can be easily viewed and edited on standard terminals.

Go Step by Step

Build your network methodically. It may be worthwhile to set up only a few remote sites initially and use them to test the features of Autopoll. Add sites to the network in groups of five or ten, eliminating problems as you go, until the complete network is installed.

Problems Do Not "Just Go Away"

In a large polling network, it is not uncommon to have problems with a few remote sites; intermittent problems are especially frustrating. Take some time to examine these difficulties carefully because they can point to problems that actually affect the entire network. Following are some questions to ask in helping to identify a problem:

◊ Are the phone lines reliable?

◊ Could fax machines, answering machines, call waiting (or other phone company services) be interfering with modems making connections?

◊ Are the modems compatible with each other?

◊ Is BLAST or BHOST being initiated correctly on the remote?

◊ Are the expected files consistently present (on both sides)?

◊ Are directory and file permissions set appropriately?

Tune BLAST Protocol Parameters

Some BLAST protocol parameters, such as the following, can be tuned for better performance with Autopoll:

Logon Timeout:	`20`
Inactivity Timeout:	`20`
DCD Loss Response:	`ABORT`

These settings permit Autopoll to react more quickly to lost connections than do the default settings. You may also wish to experiment with compression levels and packet size to find settings for best throughput. If your remote sites are running BHOST, bear in mind that the highest compression level supported by BHOST is `1` unless additional memory is allocated for compression buffers. Consult the *BHOST User Manual* for further information.

Use BPRINTER

The summary file is printed by the BLASTscript `LPRINT` command, which is tied to the BPRINTER environment variable. You can alter the behavior of `LPRINT` by changing the definition of BPRINTER in the shell environment. For example, if you set up BPRINTER to perform a concatenation as shown below, Autopoll will append summaries to an archive instead of printing them:

```
BPRINTER="cat %s >> /usr/blast/summaries"
export BPRINTER
```

BLAST substitutes the name of the short summary for the `%s` when the command is executed. For more information about BPRINTER, see "Additional Environment Variables" on page 9.

Modifying Autopoll

Because Autopoll is written in BLAST's scripting language, it is easy to customize and is thoroughly commented.

User-Supplied Scripts

The behavior of Autopoll can also be changed by writing one or more user-supplied scripts. Because Autopoll checks for the existence of these scripts at various points during execution, the scripts should be named as shown below. If Autopoll finds a user-supplied script, the script is executed by the BLASTscript `CALL` command. Autopoll tests the value of `@STATUS` when the called script returns command to Autopoll; polling continues normally if `@STATUS` equals 0; otherwise the site is marked as failed.

User-supplied scripts reside in the same directory as the Autopoll scripts. They are called at the following points during execution:

autousr0.scr	before the first site is polled (polling is aborted if this script fails).
autousr1.scr	before every attempt to `CONNECT`.
autousr2.scr	before every attempt to start `FILETRANSFER`.
autousr3.scr	before every attempt to `DISCONNECT`.
autousr4.scr	before Autopoll terminates.

Because BLASTscript variables are global, a user-supplied script must not disturb the contents of any variables needed by Autopoll. The following variables may be changed freely by any user-supplied script:

```
@STATUS           @EFERROR
@input            @temp
@xferok           @msg
@start            @filename
```

You can also create new variables if you wish. To help prevent confusion, begin new variables with "u", for example, `@uvar2`.

File I/O with User-Supplied Scripts

Autopoll opens files specified by file handles `1` through `7` at various points during execution. The handles have the following functions:

1	read-only	current site (or retry) file.
2		utility I/O.
3		utility I/O.
4		utility I/O.
5	write-only	complete polling results.
6	write-only	retry file for next cycle.
7	write-only	brief polling results (printed out).

Any of the handles reserved for utility I/O may be opened by user-supplied scripts as long as the handles are freed before the scripts return to Autopoll (i.e., each user script must close its own files). User scripts may also write to the status files, handles 5 and 7. An example of this is shown in the next section.

Autopoll closes the standard BLAST log file before calling user-supplied scripts. If a user script opens a log of its own, the log must be closed before execution returns to Autopoll.

Sample User-Supplied Script

The following user-supplied script creates a transfer command file on a remote UNIX system that contains all of the files in a given target directory. The file is retrieved for use by Autopoll in the normal way. This script provides a way to get an unspecified number of files from the remote system without sacrificing Autopoll's ability to recover from filetransfer errors.

```
# autousr2.scr
#
# Autopoll user-supplied script
#
# This script assumes that Autopoll is logged into a UNIX-based system!
#
# The script creates a command file on the remote UNIX machine
# that contains all of files in the given directory. The
# command file is then retrieved so that Autopoll can use it
# in the normal way.
#
# To create the command file, awk processes the
# output from ls. Because we will be getting each file in
# the listing, a '+' is inserted at the front of each
# filename. The /FWD switch is appended to the filename
# so the file will be deleted from the remote system if the
# transfer is successful. On the local (Autopoll) side, /OVW is
# appended to the filename so the file will overwrite an existing
# file. Assuming the target directory is /u/out and the command
# file is x.tcf, the necessary command is:
#
# ls /u/out | awk '{printf("+/u/out/%s/FWD %s/OVW\n", $1, $1)}' > x.tcf
#
# The script returns the following error codes:
#
# 0 - no error
# 1 - unable to detect remote shell prompt ($, %, or #)
# 2 - target directory is empty
# 3 - can't start BLAST on remote system
#
  set @upath = "/u/out"                      # adjust path to suit!
  set @temp = "ls "
  strcat @temp, @upath
  strcat @temp, " | "                        # e.g, "ls /u/out | "
  strcat @temp, "awk ‘{printf(\"+"           # attach awk command
  strcat @temp, @upath
  strcat @temp, "/%s/FWD %s/OVW\\n\", $1, $1)}' > "
  strcat @temp, @worktcf                   # @worktcf is Autopoll variable
  tsend cr                                 # get the attention of the remote
  ttrap 2, "$", "#", "%"
  if @STATUS = "0" return 1                # return error
  wait 1 idle
```

```
  tsend "ls ", @upath, " | wc -l", cr       # how many files?
  ttrap 2, " 0"                             # no files in directory?
  if @STATUS = "1"
    fwrite 5, @b, " Remote directory ", @upath, " is empty."
    write "remote directory ", @upath, " is empty!"
    return 2                                  # dir is empty, nothing to do
  end
  write "preparing command file remotely"     # inform user
  wait 1 idle
  tsend @temp, cr                             # run the awk script
  ttrap 2, "$", "#", "%"
  if @STATUS = "0" return 1
  filetransfer                                # now retrieve the command file
  if @STATUS not = "0" return 3               # can't start BLAST on remote
  get
  @worktcf
  @worktcf
  to
  esc
  return @EFERROR                             # returns 0 if no error
#
# End of script.
```

Configuration Worksheets

The following worksheets may help you organize the large amount of information needed to set up a polling network successfully.

A. List Machines

List the machines in your polling network. For completeness, include information for the central site as well.

Site	Name	Phone	Modem Type	Port	BLAST Version	System Type
Central						
1.						
2.						
3.						

B. Decide on Setups

Decide whether or not different setup files will be needed for each site. If so, create the setups and list their names. Remember, Autopoll loads the setup autopoll.su by default.

Site	Name	Setup Name
1.		
2.		
3.		

C. Set Up the Remote Sites

Set up the remote sites and test each connection manually. Make sure the following sequence of keyboard commands work flawlessly:

Connect	dials the modem and logs in if necessary.
Filetransfer	enters BLAST filetransfer.
ESC	exits BLAST filetransfer.
Disconnect	logs off and hangs up the phone.

D. Create the Site File

Build the entries in the site file with any standard text editor, selecting appropriate name(s) for the TCF files.

site filename: ___________.dat

Setup	Name	Phone	Baud	TCF

E. Create the Transfer Command Files

List the files to be transferred to and from each site and the direction of transfer (S=`SEND`, G=`GET`). Afterward, write the various TCF files and put them in the autopoll directory.

Site	S/G	Remote Name	Local Name	Options
1.				
2.				
3.				

F. Decide on Cycles

Decide how many cycles to allow for polling and when to start:

Cycles:

Start time:

G. Build the Command Line to Start Autopoll

Use the following format:

```
blast autopoll -sautopoll max_cycles site_file [start_time]
```

H. Check Environment Variables

Check the values of BLASTDIR, SETUPDIR, PATH, and BPRINTER. When they are correct, change to the autopoll directory, type in the command line, and let Autopoll take over!

Appendix F

PAD Parameters

The X.3 standard specifies a set of parameters defining how a PAD is to perform its task of assembling and disassembling the data stream. Each parameter is identified by a number and has several optional values. For example, Parameter 2 specifies whether or not the PAD is to echo input characters. A value of 0 specifies no echo, and a value of 1 specifies echo. This parameter can be set manually from the terminal in the form Parameter 2 = 0, or, in some cases, the parameters can be downloaded automatically from the X.25 host system to the PAD.

In the following discussion of the parameters relevant to BLAST operation, the word "must" refers to critical settings while "should" refers to non-critical ones. "DTE" (Data Terminal Equipment) refers to the BLAST terminal or computer that is generating the data stream being processed by the PAD.

Parameter 1 Escape to Command Level

0 = Escape not possible
1 = Escape possible (default)

This parameter allows an escape to command level. If escape is enabled, the occurrence in the terminal data stream of two carriage returns (CR) in the sequence "CR@CR" will cause the PAD to go into command mode; this sequence is similar to the AT "+++" sequence. Because BLAST encoding does not use the "@" character, the setting of this parameter is irrelevant.

Parameter 2 Echo

0 = No echo
1 = Echo (default)

This parameter specifies whether or not the PAD echoes input characters. This parameter must be set to 0 (no echo) for BLAST operation.

Parameter 3 Data Forwarding Character(s)

NOTE: Values may be combined with the or operator

0 = No data forwarding character
1 = Alphanumerics
2 = Carriage Return [CR] (default)
4 = Escape [ESC]
8 = Editing characters
16 = Terminators
32 = Form effectors
64 = Control characters

This parameter specifies the character(s) that will trigger the PAD to transmit all currently accumulated data as a packet. Because BLAST appends a CR to each packet, BLAST's efficiency over X.25 networks is greatly improved if "2" is the setting for Parameter 3.

Parameter 4 Idle Timer

0 = Timer disabled
N = Multiples of .05 seconds (default = 80 [4 secs])

This parameter enables the PAD to transmit all currently accumulated characters as a packet if the interval between successive characters received from the terminal exceeds the specified Idle Timer delay. This parameter does not normally affect BLAST operation unless the parameter is set to an extremely small value. Such a setting could cause the PAD to send an incomplete BLAST packet if the BLAST computer pauses momentarily.

Parameter 5 XON/XOFF Flow Control of DTE by the PAD

0 = PAD may not exert flow control (default)
1 = PAD may exert flow control

This parameter specifies whether or not the PAD can exert flow control. Under heavy network traffic conditions, a PAD may not always be able to keep pace with the incoming data stream, in which case it is preferable to exert flow control on the DTE. If the PAD is not allowed to exert flow control, it will occasionally drop incoming characters (see "XON/XOFF Pacing" on page 31). Because BLAST encoding does not use control characters, including the CTRL S and CTRL Q flow control characters, it is compatible with XON/XOFF flow control by the PAD.

Unfortunately, some PADs are not intelligent in their use of flow control, generating XON/XOFF sequences as often as every five characters. This frequent generation of XON/XOFF sequences significantly reduces BLAST throughput and increases the possibility that an XON or XOFF will be lost. BLAST can be set to unilaterally resume transmission after a fixed delay period (typically 30 seconds) in the event that an XON from the PAD is lost; however, it is not desirable to rely on this mechanism.

Because BLAST is an error-free protocol, it compensates for lost characters through retransmission of data blocks. If this is an occasional occurrence, it may be preferable to disable PAD to DTE flow control. On the other hand, if the PAD is very heavily loaded and/or the PAD uses XON/XOFF intelligently, it is better to enable flow control.

The XON/XOFF setting of the computer running BLAST should always match that of the PAD.

Parameter 6 Suppression of Service Signals

0 = Messages not sent
1 = Messages sent (default)

Must be set to 0.

Parameter 7 Break Options

0 = Do nothing (default)
1 = Send interrupt packet to host
2 = Send reset packet to host
8 = Escape to PAD command state
21 = Flush

Should be set to 0.

Parameter 8 Discard Output

0 = Normal data delivery (default)
1 = Discard all output to DTE

Must be set to 0.

Parameter 9 Carriage Return Padding

0 = No padding (default)
1–31 = Character delay times

Should be set to 0.

Parameter 10 Line Folding

0 = No line folding (default)
N = Characters per line before folding

This parameter specifies if and how often the PAD is to insert a carriage return and line feed automatically to break long text lines into shorter ones. It must be set to 0.

Parameter 11 Binary Speed

0 = 110 bps
↓
18 = 64000 bps

This parameter is transparent to BLAST.

Parameter 12 XON/XOFF Flow Control of PAD by the DTE

0 = DTE may not exert flow control (default)
1 = DTE may exert flow control

See discussion under Parameter 5.

Parameter 13 Linefeed (LF) Insertion

0 = No linefeed insertion (default)
1 = Insert LF after CR on output to DTE
2 = Insert LF after CR on input from DTE
4 = Insert LF after CR on echo to DTE

Should be set to 0.

Parameter 14 Linefeed Padding

0 = No padding (default)
1–15 = Number of null characters

Should be set to 0.

Parameter 15 Editing

0 = Editing disabled (default)
1 = Editing enabled

This parameter enables local editing of text within the PAD before transmission through the network. If editing is enabled, transmission of the timer is disabled. Must be set to 0.

Parameters 16–18 Editing Options

[Does not apply if Parameter 15 = 0, editing disabled.]

Parameter 19 Editing PAD Service Signals

[Does not apply if Parameter 6 = 0, service signals disabled.]

Parameter 20 Echo Mask

[Does not apply if Parameter 2 = 0, no echo.]

Parameter 21 Parity

0	=	No parity checking or generation (default)
1	=	Check parity only
12	=	Parity generation only
13	=	Both parity checking and generation

Should be set to 0.

Parameter 22 Page Wait

0	=	Page wait disabled (default)
1–255	=	Wait after the specified number of lines are displayed

Must be set to 0.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

TO: BLAST Technical Support **FAX #: 919-542-0161**

FROM: **Voice #:**

COMPANY: **FAX #:**

DATE:

I**MPORTANT:** Please provide us with the following information:

Your BLAST version #_________________ Serial #________________________

Your operating system_________________ Version #______________________

Where does the problem occur? (please circle)

Installation Filetransfer Terminal Emulation Scripting

Background Remote Control Other__________________________

Please describe the problem:

How Was It?

We would like to hear your feedback on the usefulness of this document. Your opinions can help us improve it in the future.

BLAST Professional UNIX User Manual   2MNUNIX   October 1999

1. Please rate the following:

	Excellent	Good	Fair
Ease of finding information			
Clarity			
Completeness			
Accuracy			
Organization			
Appearance			
Examples			
Illustrations			
Overall satisfaction			

2. Please check areas that could be improved:

- ☐ Introduction
- ☐ Organization
- ☐ Include more figures
- ☐ Include more examples
- ☐ Add more detail
- ☐ More step-by-step procedures
- ☐ Make it more concise
- ☐ Make it less technical
- ☐ More quick reference aids
- ☐ Improve the index

3. Please elaborate on specific concerns and feel free to comment on any topics not raised previously:

Please FAX or mail these comments to us. Our contact information is listed on the title page of this manual. Thank you for your input.