

BLAST[®]

Professional VMS User Manual

BLAST[®]
SOFTWARE

The File Transfer Authority

©2001 by BLAST, Inc.
49 Salisbury Street West
Pittsboro, NC 27312
All Rights Reserved

Manual #2MNPVMS
10/01

The information in this manual has been compiled with care, but BLAST, Inc., makes no warranties as to accurateness or completeness, as the software described herein may be changed or enhanced from time to time. This information does not constitute commitments or representations by BLAST, Inc., and is subject to change without notice.

BLAST® is a registered trademark, and BLAST Professional™, BLAST Professional VMS™ and TrueTerm™ are trademarks of BLAST, Inc. Any trademarks, trade-names, service marks, service names owned or registered by any other company and used in this manual are proprietary to that company.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (b) (3) (ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013.

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>

© Copyright 2001 by BLAST, Inc.

Table of Contents

1 Introduction 1

BLAST Software Registration	1
The BLAST Package	2
BLAST Professional Features.	2
How to Use This Manual	3
Comments and Suggestions	4
BLAST Technical Support	5

2 The BLAST Environment 7

Introduction.	7
Assigning Symbol Values.	7
Command Line Switches	10
Communications Ports	14
Flow Control.	19

3 BLAST Quickstart 23

Starting BLAST	23
The BLAST Screen.	24
Three Keys to Remember	26
The BLAST Menus.	27
A Quickstart File Transfer	28

4 The Menus **35**

Moving Through the Menus	35
Frequently Used Keys.	36
The Offline Menu	37
The Online Menu	39
The Filetransfer Menu.	40
The Local Menu	41
The Remote Menu.	43
Automation with BLASTscript.	44

5 The Setup **45**

What is a Setup?	45
Setup Fields.	48
BLAST Protocol Subwindow	58
Kermit Protocol Subwindow.	63
Xmodem and Ymodem Protocol Subwindow	67
Zmodem Protocol Subwindow	69
File Attributes Subwindow	73

6 BLAST Session Protocol **77**

What is a Protocol?	77
The BLAST Session Protocol.	78
BLAST Protocol Design.	79
Starting a BLAST Session	81

Ending a BLAST Session	84
Performing Filetransfer Commands	85
Transfer Command File	94
BLAST Protocol Remote Menu	97
Automating the BLAST Session Protocol	98
Fine-Tuning the BLAST Session Protocol	98
Filetransfer Security with BLAST Protocol	100

7 Kermit Protocol 103

Kermit Filetransfer Menu	103
Sending and Receiving Files with Kermit	104
Kermit Remote Menu	109

8 Xmodem, Ymodem, and Zmodem Protocols 111

Command Line Features	112
Xmodem Protocol	112
Ymodem Protocol	116
Zmodem Protocol	118

9 Text Transfers 121

Introduction	121
Uploading Text to a Remote Computer	121
Downloading Text from a Remote Computer	123

10 Secure BLAST

125

Securing Your System	125
VMS Tools	125
Using Secure BLAST	127
BLPASSWORD	128
BLSECURE	135
SECURE	139
Using the Password	140

11 Introduction To Scripting

143

Starting Out	143
Learn Mode	148

12 BLASTscript Topics

153

Scripting Basics	153
Manipulating Text	158
Managing the Screen Display	162
Communicating with Other Programs	164
File Transfers with BLAST Protocol	165
File Transfers with Kermit	168
File Transfers with Xmodem and Xmodem1K	171
File Transfers with Ymodem and Ymodem G	172
File Transfers with Zmodem	174
BLAST Operation as a Pseudohost	176

Using Log Files for Error Checking	177
Text Transfers	178

13 Connecting and Disconnecting **181**

Introduction	181
BLASTscript Libraries	181
The Index Utility	186

14 BLASTscript Command Reference **189**

Introduction	189
Data Types	189
Syntax Rules	192
Commands That Set @STATUS	193
Manipulation of Binary Data	193
BLASTscript Commands	194

15 BLASTscript Reserved Variables **233**

16 Data Stream Control **273**

Introduction	273
Data Stream Filtering and Alteration	273
Standard BLAST Terminals	277

17 Limited Control of a Remote PC **281**

Connecting to the Host PC 281
Transferring Files to and from the Host PC 283
Disconnecting from the Host PC 284
Using Terminal Mode 284
Modifying BHOST Settings 285

Appendix A Error Messages **293**

Introduction. 293
BLAST Protocol Functions. 293
Transfer File Management 294
Utility File Management 295
Scripting 295
Command File Processing 296
Initialization 296
Script Processor 297
Network. 297

Appendix B Key Definition Charts **299**

Setup Keys 299
Attention Key Sequences 300
Hot Keys 300

Appendix C Troubleshooting **301**

Appendix D The ASCII Character Set 303

Appendix E Autopoll 305

The Autopoll Script 305

Installing Autopoll 306

Starting Autopoll. 306

The Site File 308

Transfer Command File 309

Overview of Autopoll Script Actions 310

Configuration Example. 311

Other Files Using the Filename Stub 313

Tips and Tricks 316

Modifying Autopoll 317

Configuration Worksheets 321

Appendix F PAD Parameters 323

INDEX 329

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, and ISDN Terminal Adaptors.
- ◇ Hardwired RS-232 connections.

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.

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!

BLAST VMS supports TTY and PASSTHRU modes, which ensure complete and accurate transmission of control characters to the terminal hardware.

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.

- ◇ 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 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 annually or on a per-incident basis. 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 system information, type on the command line:

```
show system
```

On many systems, you may display just the version number by typing on the command line:

```
show system /noprocess
```

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 symbols 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.

Assigning Symbol Values

When BLAST is installed, by default all BLAST files are placed in one of two directories:

- ◇ A directory containing executable files, the online help file, and both system and modem libraries. The `BLASTDIR` symbol defines the directory containing these files. *BLASTDIR must exist in order for BLAST to execute.*

- ◇ A directory containing support files, which includes setup files, autopoll scripts, and example files. The **SETUPDIR** symbol defines the directory containing these files. If no **SETUPDIR** exists, **BLAST** will look in **BLASTDIR** for setup files. User-supplied script files should be stored in this directory or in the current directory.

BLAST uses symbol values from the symbol table to locate files needed to run **BLAST**. During installation, **BLAST** creates **BLASTSYM.COM**, a text file containing DCL commands that define symbols needed to run **BLAST**. This file can be called from the system login command file (**SYLOGIN.COM**) by adding the following line to **SYLOGIN.COM**:

```
$SYS$MANAGER:BLASTSYM.COM
```

If you want to limit access, you may include this command in your individual login command file rather than **SYLOGIN.COM**. Additional lines can be added to **BLASTSYM.COM** to define other symbols. For example, symbol definitions can be added to allow you to run the following executables: **BLPASSWD.EXE**, **INDEX.EXE**, **BLSECURE.EXE**, and **SECURE.EXE**. Note that if you move **BLAST** files, you should modify the symbol values in **BLASTSYM.COM** to reflect the new location of the files.

BLAST Symbols

Following is a description of the syntax for defining symbols as well as possible symbol values; default values are indicated in bold brackets.

\$BANNERTIME:==*delay* 0 – 99

where *delay* is the time in seconds that the initial screen is displayed. The default display time for the banner is 4 seconds.

EXAMPLE:

```
$BANNERTIME:==2
```

\$BLAST:==\$*path*BLAST.EXE **[SYS\$SYSTEM:BLAST.EXE]**

where *path* is the path of the directory containing the **BLAST** executable, **BLAST.EXE**.

EXAMPLE:

```
$BLAST:==$SYS$SYSDEVICE:[USER.JOE.BLAST]BLAST.EXE
```

\$BLASTDIR:==*path* [SYS\$SYSTEM:[BLAST]]

where *path* is the path of 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:

```
$BLASTDIR:==$SYS$SYSDEVICE[USER.JOE.BLAST]
```

\$BLPASSWD:==\$*path*BLPASSWD.EXE

where *path* is the path of the directory containing the executable BLPASSWD.EXE. The default directory for BLPASSWD.EXE is SYS\$SYSTEM.

EXAMPLE:

```
$BLPASSWD:==$SYS$SYSDEVICE:[USER.JOE.BLAST]BLPASSWD.EXE
```

\$BLSECURE:==\$*path*BLSECURE.EXE

where *path* is the path of the directory containing the executable BLSECURE.EXE. The default directory for BLSECURE.EXE is SYS\$SYSTEM.

EXAMPLE:

```
$BLSECURE:==$SYS$SYSDEVICE:[USER.JOE.BLAST]BLSECURE.EXE
```

\$BPRINTER:==*device*

where *device* is the target for printer output. The default target for printer output is SYS\$SYS\$PRINT.

EXAMPLE:

```
$BPRINTER:==$SYS$SYS$PRINT
```

\$EDITOR:==\$filename

where *filename* is the name of the editor program that will be invoked by the Edit command from the Local menu. The default editor is EDIT.

EXAMPLE:

```
$EDITOR:==LSE
```

\$INDEX:==\$pathINDEX.EXE

where *path* is the path of the directory containing the executable INDEX.EXE. The default path for INDEX.EXE is SYS\$SYSTEM.

EXAMPLE:

```
$INDEX:==$SYS$SYSDEVICE:[USER.JOE.BLAST]INDEX.EXE
```

\$SECURE:==\$pathSECURE.EXE

where *path* is the path of the directory containing the executable SECURE.EXE. The default directory for SECURE.EXE is SYS\$SYSTEM.

EXAMPLE:

```
$SECURE:==$SYS$SYSDEVICE:[USER.JOE.BLAST]SECURE.EXE
```

\$SETUPDIR:==\$path [SYS\$SYSDEVICE:[BLAST]]

where *path* is the path of the directory in which the BLAST setup files are stored.

EXAMPLE:

```
$SETUPDIR:==SYS$SYSDEVICE:[USER.JOE.BLAST]
```

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:

`blast [setupname] [-sscriptname] [argument] [-b] [-dd] [-dt] [-e] [-h] [-n] [-p] [-q] [-v or -?] [-x] [-y]`

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, `-sscriptname` 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`.

-sscriptname

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.

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 in 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.

-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.

-dd

Changes the default date format globally (see @DATEFORMAT on page 237). 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"
```

-enumber

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 8 for more information on Xmodem and Ymodem.

EOT timeout for Xmodem and Ymodem may also be specified with the BLASTscript reserved variable @XYEOT.

-h

Executes BLAST in host mode. In host mode, BLAST runs in File-transfer 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
```

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.

Using the `-h` switch, BLAST can perform X, Y, or Zmodem file transfers. See “BLAST Operation as a Pseudohost” on page 176.

-n

Forces BLAST to execute in no display mode. Displays may be selectively reenabled through BLASTscript commands (see “Managing the Screen Display” on page 162). This switch allows you to integrate BLAST into your applications without losing the information previously written to the screen.

-px

Specifies the pad character (`x`), expressed as a decimal value, to be used with Xmodem transmissions. See Chapter 8 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.

-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.

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:

```
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.

Communications Ports

BLAST establishes a communications session with asynchronous serial ports. The port used by BLAST is specified in the Connection setup field (page 51) or in the BLASTscript reserved variable @COMMPORT (page 236).

In addition to serial ports, devices such as multi-port serial boards and X.25 PADs permit software, like BLAST, to access the hard-

ware. If the manufacturer of a device does not provide a standard asynchronous interface, BLAST cannot open the device. If RS-232 capabilities are not correctly implemented in the device, those capabilities 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.

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. 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 73) or the reserved variable @PAKTSZ (page 253) specifies the number of bytes of data that BLAST will transmit in each BLAST packet. This specification does not include any bytes asso-

ciated 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:

$$\text{BLAST Packet} = \frac{[(\text{X.25 Frame} - 4) \times 7] - 9}{8}$$

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 220.

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

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 *PAD Parameters* on page 323 for a complete explanation of PAD Parameters.

Checking Port Permissions

If you do not know the name of the port you want to access, consult your system administrator for the name.

The port must have read and write permission in order for BLAST to access it. You can check the device protection of a port by issuing the following command:

```
show device device_name /full
```

where *device_name* is the port name. For example, to check the device protection for TTA1:, you would type the following:

```
show device TTA1: /full
```

You should see output similar to the following:

```
Terminal TTA1:, device type unknown, is online, record-oriented device, carriage control.

Error count           0      Operations completed      128613
Owner process         ""      Owner UIC                 [SYSTEM]
Owner process ID      00000000  Dev Prot                 S:RW,O:RW,G:RW,W
Reference count       0      Default buffer size      80
```

The device protection output “S:RW,O:RW,G:RW,W” indicates that the port has universal read and write permissions.

If the port does not have these permissions, you can change the device protection for the port if your permissions allow, by issuing the following command:

```
set protection=(s:rw,o:rw,g:rw,w) /device device_name
```

where *device_name* is the port name.

Automatic 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 setup field. 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 defined by the BLASTDIR symbol.
- ◇ tests each listed port in the order specified until an available port 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 52).

baud_rate specifies the baud rate in the same format used for the Baud Rate setup field (page 53).

IMPORTANT: The hunt file, including the beginning and end of the file, may not contain any extra spaces or lines.

For example, BLAST would test the devices DUA0: and DUA2: listed in the following hunt file:

```
1 DUA0: MICROCOM 19.2
0 DUA1: USRCour 9600
1 DUA2: Intel 9600
```

BLAST would ignore the entry for DUA1: because it is preceded by a “0”.

Port Parameters for BLAST in Host Mode

Serial ports 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 port parameters accordingly when you go online.

When someone logs into your VMS system, the system sets serial port parameters according to your TERMINAL. When BLAST is run in host mode on your 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 your port settings. For instance you may want to change your flow control settings to RTS/CTS. To do this, you would issue the following command at the system command prompt:

```
set TERM device_name /nohostsync/noTTsync/modem/permanent
```

where *device_name* is the port name. Note that you must have proper permissions to make this change. See your system documentation for details.

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 port 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 54) and the reserved variable @RTSCTS (page 258) 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 RTS/CTS Pacing setup field is set to YES, BLAST sets the serial port to use RTS/CTS flow control; when it is set to NO, BLAST disables RTS/CTS flow control.

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.

XON/XOFF Pacing

The XON/XOFF Pacing setup field (page 53) and reserved variable @XONXOFF (page 266) 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 XON/XOFF Pacing setup field is set to YES, BLAST attempts to set the serial port to use XON/XOFF flow control; when it is set to NO, BLAST 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 serial port, modems, 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 port will issue an XOFF when the 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 serial port will issue an XOFF when the 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 port 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 184).

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.
- ◇ Define all the necessary BLAST symbols (see “Assigning Symbol Values” on page 7 and your system documentation for more information).
- ◇ 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.

The Offline menu (Figure 3-1) will appear or, if this is the first time that BLAST has been run, the Online Help screen will appear. To move from the Online Help screen to the Offline menu, press *CANCEL*. From the Offline menu, you can control BLAST interactively.

The BLAST Screen

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

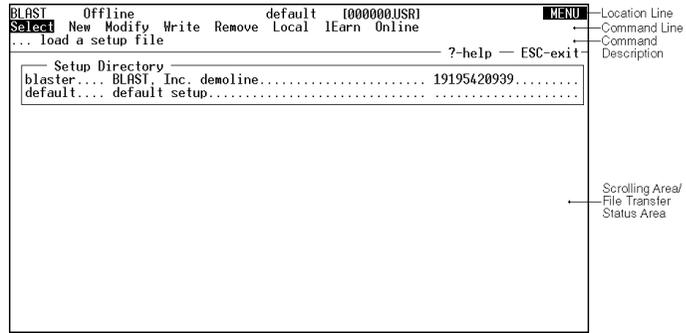


FIGURE 3-1

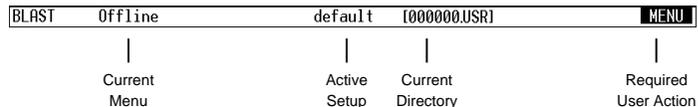
Command Area

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 26).

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 selected for the 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.

```
BLAST Filetransfer default [000000.JSR] 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>
-----<< Entering BLAST Transfer Mode >>-----
**** MacBLAST 10.2 on remote system luovml ****
```

FIGURE 3-3

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 *ATTN* to abort script operations or initiate other special key combinations. Press *ATTN ATTN* to return to the Online menu from Terminal mode. (The *ATTN* key can be redefined; see the discussion of the Attention Key setup field on page 55.)

CANCEL To cancel the current action, return to the previous menu, or exit BLAST, press CTRL A when working at the console and press ESC or CTRL A when working at a terminal.

HELP While in Terminal mode, press *ATTN H*; when not in Terminal mode, press ?

The BLAST Menus

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

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.

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

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 on the next page.

```

BLAST      Offline          BLASTER      [USER.TEST]      MENU
... ^E-up ^X-down ^R-first ^C-last ^P-prompt ^T-clear ^A-exit
... press ^T to clear or enter new text

----- ?-help -- ^A-exit -----
- Setup for: BLASTER
  Description: BLAST, Inc. demoline
  Phone Number: 19195420939
  System Type: UNIX
  Userid: reliable
  Password: XXXXXXXXXXXXXXXX
  Attention Key: ^K_
  Connection: TTA1:
  Emulation: TTY
  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_

  Script File:
  Protocol: BLAST...
  Log File:
  Packet Size: 256_
  Translate File:
  File Attributes: ...

```

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: TTY
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 **TTY**. 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.

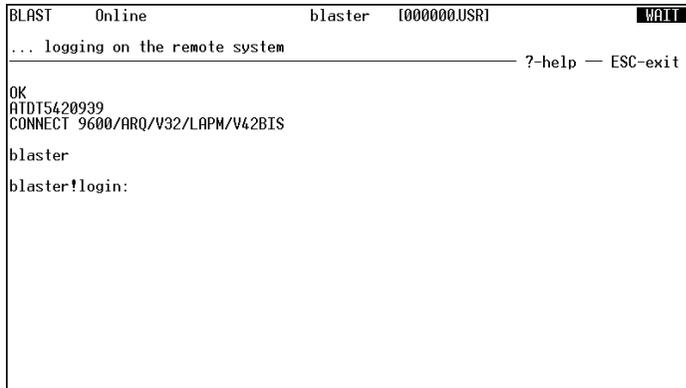
Connecting to Blaster

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).



```
BLAST Online blaster [000000.USR] WAIT
... logging on the remote system
----- ?-help - ESC-exit
OK
ATDT5420939
CONNECT 9600/ARQ/V32/LAPM/V42BIS
blaster
blaster!login:
```

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).

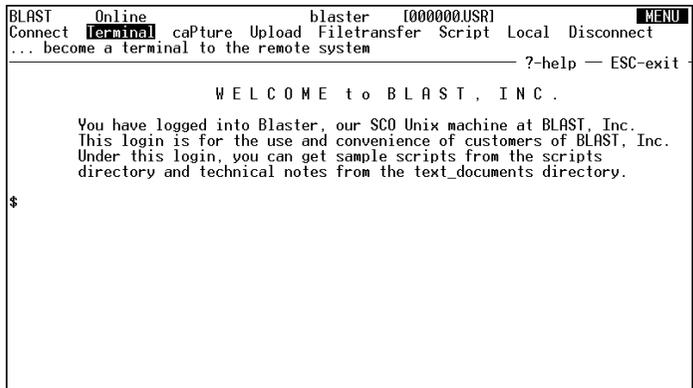
Blaster assumes that a dial-in user will be using a VT-100 terminal. If you are using a VT terminal or your console operates as a VT or ANSI terminal, you should not have problems. Problems such as the screen not clearing, improper positioning of characters, and strange character sequences indicate that 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 have logged into Blaster using a WYSE 60 terminal, enter Terminal mode by choosing Terminal from the Online menu and then at the command line type the following:

```
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 via a telnet session window and experience the problems described above, you may be defaulting to an emulation other than VT100. For information on resetting your console or terminal, consult your system, telnet-application, or hardware documentation.



```
BLAST Online blaster (000000USR) MENU
Connect Terminal caPture Upload Filetransfer Script Local Disconnect
... become a terminal to the remote system ?-help - ESC-exit

WELCOME to BLAST, INC.

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, next page).

```

BLAST Filetransfer blaster [000000USR]
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 [uovl] ****

```

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:

BLAST is asking for the name that will be given to the file when it is received on the local computer. 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 Blaster

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`, `BACKSPACE`, or the cursor keys.

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 45.

Frequently Used Keys

The most frequently used keys when running BLAST are:

- 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 the discussion of the Attention Key setup field on page 55).
- CANCEL* To cancel the current action, return to the previous menu, or exit BLAST, press ESC or CTRL A.
- HELP* While in Terminal mode, press *ATTN* H; 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 55) or by setting the BLASTscript reserved variable @ATTKEY (page 234).

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

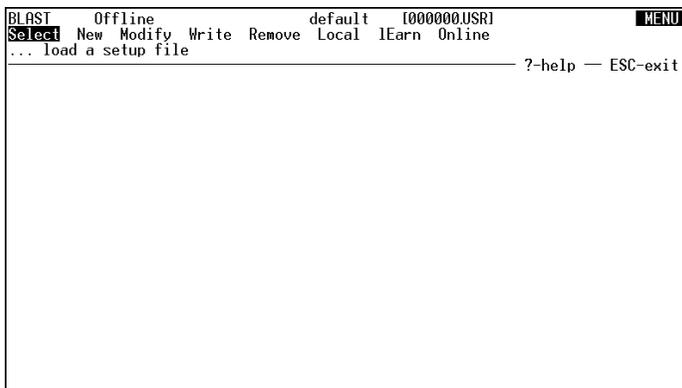
The *CANCEL* key is used to cancel the current action. It also returns to a previous menu from a lower level menu and is used to exit BLAST from the Offline menu. The exception to this rule is that you must press *ATTN ATTN* to escape from Terminal mode.

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.

The Offline Menu

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 24 for a description of these sections. We will be concerned here primarily with the Command Area, specifically the Command Line.



```
BLAST Offline default [000000.USR] MENU
Select New Modify Write Remove Local LEarn Online
... load a setup file
?-help - ESC-exit
```

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 45 for 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 41).

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 148 for more details.

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

Online – Opens the communication port and 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. 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 55). See “Translate File Format” on page 274 for more information on translate files.

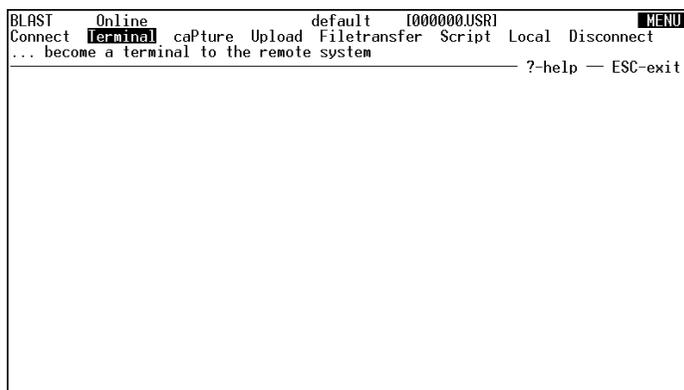


FIGURE 4-2

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 menu. See “Standard BLAST Terminals” on page 277.

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 123 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 121 for further information.

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 11–15 for information on scripts.

Local – Allows you to perform local system commands. This command takes you to the Local menu described on page 41.

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 on the next page.

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 43; see also “Kermit Remote Menu” on page 109.

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 59) 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 94, “Transfer Command File” on page 309, and “Transfer Command Files” on page 312).

```
BLAST Filetransfer default [000000]USR1 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>
-----<< Entering BLAST Transfer Mode >>-----
**** MacBLAST 10.2 on remote system [uovm] ****
```

FIGURE 4-3

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.

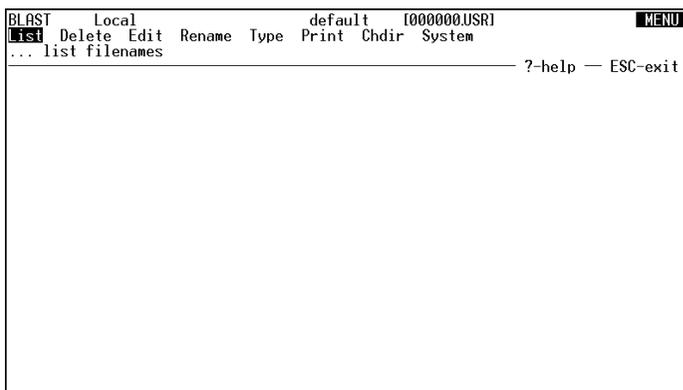


FIGURE 4-4

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 defined by the EDITOR symbol (see EDITOR on page 10).

Rename – Renames a local file.

Type – Displays a local file in the scrolling area.

Print – Prints a file to the local printer or print spooler as defined by the BPRINTER symbol (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 `logout` 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 or Kermit protocol, the Filetransfer 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.

```
BLAST Remote default [000000.USRI] MENU
list Delete Rename Type Print Chdir More
... list remote filenames
_ local _ opt - % xfer - file size - byte cnt - ln qual -
$: <idle>
R: <message> good (00)
? - help - ESC - exit

s.
s..
swinblast.exe
shb_contl.dll
shb_comdr.dll
shb_async.dll
stest.scr
shb_filei.dll
shb_ftran.dll
shb_bprot.dll
shb_zmodm.dll
shb_compr.dll
shb_term1.dll
shb_vtenu.dll
svt_font.fon
use More to continue...
```

FIGURE 4-5

Figure 4-5 above shows the BLAST protocol Remote menu. The commands of this menu, which differ from the Kermit Remote menu, are described briefly below. For a fuller discussion of the commands of the Remote menus, see “BLAST Protocol Remote Menu” on page 97 and “Kermit Remote Menu” on page 109.

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 11–15 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 28). 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 cursor keys or the keys listed at the top of the Command Area to highlight the setup that you want to load. After highlighting a setup, press ENTER to load the selected setup.

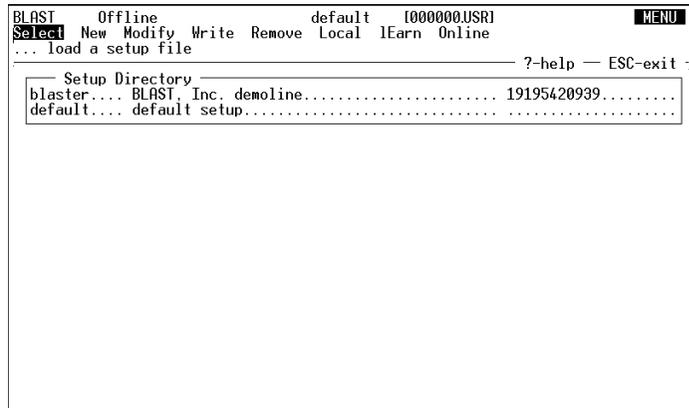


FIGURE 5-1

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. You may want to use the location of the remote site as the setup name, or some other easily remembered 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:

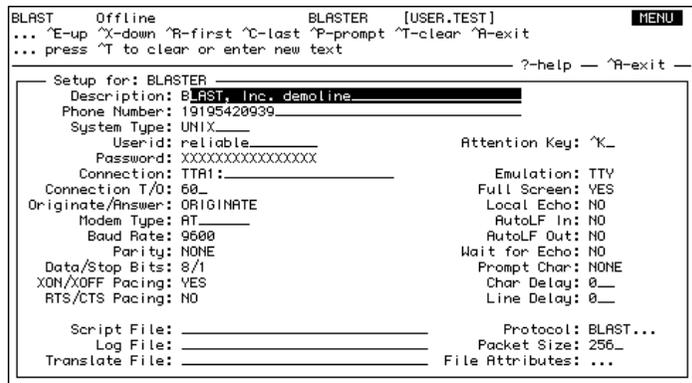


FIGURE 5-2

A field must be highlighted before you can modify its value. Use the cursor keys or the keys listed at the top of the Command Area 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, such as Phone Number, require user input. There are two modes for entering information into these fields, “Select” mode and “Edit” mode. In “Select” mode, the left and right arrow keys move the cursor to another field; in “Edit” mode, the arrow keys move the cursor within the field. To enter “Select” mode, simply begin typing; what you type will overwrite the entire field. To enter “Edit” mode, press *CTRL P*; the cursor will be positioned at the end of the field. Control keys for both modes are as follows:

<u>Key</u>	<u>Command</u>
CTRL V	Toggle between Insert and Overwrite mode.
CTRL F	Move to the beginning of the line.
CTRL G	Move to the end of the line.
BACKSPACE	Delete
CTRL T	Clear field

To toggle between “Edit” and “Select” mode, press CTRL P. Once you have finished editing a field, press ENTER to move to the next field or alternatively, if you are in “Select” mode, use the arrow keys to select another field.

The File Attributes field and some 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 15). The Online Help for each field also

contains detailed information. The individual fields are discussed on the pages listed in the following table:

<u>FIELD</u>	<u>PAGE</u>	<u>FIELD</u>	<u>PAGE</u>
DESCRIPTION:	49	ATTENTION KEY:	55
PHONE NUMBER:	49	EMULATION:	55
SYSTEM TYPE:	49	FULL SCREEN:	55
USERID:	50	LOCAL ECHO:	56
PASSWORD:	51	AUTOLF IN:	56
CONNECTION:	51	AUTOLF OUT:	56
CONNECTION T/O:	51	WAIT FOR ECHO:	56
ORIGINATE/ANSWER:	51	PROMPT CHAR:	57
MODEM TYPE:	52	CHAR DELAY:	57
BAUD RATE:	53	LINE DELAY:	57
PARITY:	53	PROTOCOL:	58
DATA/STOP BITS:	53	BLAST SUBWINDOW:	58
XON/XOFF PACING:	53	KERMIT SUBWINDOW:	63
RTS/CTS PACING:	54	X/YMODEM SUBWINDOW:	67
SCRIPT FILE:	54	ZMODEM SUBWINDOW:	69
LOG FILE:	54	PACKET SIZE:	73
TRANSLATE FILE:	55	FILE ATTRIBUTES SUBWINDOW:	73

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 1024 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 Alpha or 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 for the system. See Chapter 13 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.

BLASTscript variable: @PASSWORD

Connection any valid device name
hunt filename NONE

Specifies the communications port or hunt file that BLAST will use for the current session. Valid options are:

Device name – Any valid asynchronous port (e.g., `ttal:`).

Hunt filename – The name (including path) of a hunt file that lists available devices, preceded by the “<” character. Refer to “Automatic Port Searching” on page 17 for details concerning hunt files.

NONE – Entering NONE in the connection field allows the user to run scripts without opening a communications port. If you enter NONE into the field, the only command options available from the Online menu will be Script and Local; furthermore, any BLAST-script commands requiring an open communications port, such as FILETRANSFER and TERMINAL, will not be allowed.

BLASTscript variable: @COMMPORT

Connection T/O 0 – 999 [60]

This feature has not been implemented in this version of BLAST.

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 – Megahertz 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
USR32 – 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 for the modem. See Chapter 13 for more details on MODEMS.SCR and user-defined modem scripts.

BLASTscript variable: @MODEM

BLAST will resume transfer. See “Port Parameters for BLAST in Host Mode” on page 18.

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 “Port Parameters for BLAST in Host Mode” on page 18.

BLASTscript variable: @RTSCTS

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 266.

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 274 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. For example, 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. 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

[TTY] PASSTHRU

The terminal emulation values are PASSTHRU, in which the characters received by the serial port are displayed without change, and TTY.

BLASTscript variable: @EMULATE

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 non-zero 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

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 5-3:

FIGURE 5-3

```
BLAST Protocol
Logon T/O: 120
Inactivity T/O: 120
7 Bit Channel: NO
Window Size: 16
DCD Loss Response: IGNORE
Use "A" Protocol: NO
Filtering: OFF
Retransmit timer: 4

ACK Request Frequency: 4
Number of Disconnect Blocks: 3
Launch String: \r
Transfer Password:
Enable /FWD and /STR: YES
Enable /DWH and Remote Cnds: YES
Send Compression Level: 4
Receive Compression Level: 4
Append VMS File Switches: NO
```

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 “The

BLAST Session Protocol” on page 78 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 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 59).

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 file-name 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 100).

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

Append VMS File Switches

[NO] YES

For BLAST protocol, appends all relevant VMS-specific file transfer switches (see “VMS-Specific Switches with BLAST Protocol” on page 91) to the names of files sent to another VMS system.

IMPORTANT: If you set this field to YES for transfers to a system other than VMS, you may get an error message.

BLASTscript variable: @VMSFILESW

Kermit Protocol Subwindow

Selecting KERMIT and pressing ENTER displays the subwindow shown in Figure 5-4 below:

FIGURE 5-4

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
Send Packet Size: 90	Receive Packet Size: 90
Pad Character: ^@	Pad Character: ^@
Padding: 0	Padding: 0
Timeout (seconds): 10	Timeout (seconds): 10
Filename Conversion: NO	Filename Conversion: NO
Transfer Type: BINARY	Transfer Type: BINARY
Retry Limit: 10	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. Note that the re-

remote Kermit server's Receive Pad Character should also be set to this value.

BLASTscript variable: @KSPADCH

For receiving files with Kermit: specifies an alternate character to pad each packet received by the local computer. Note that the remote Kermit server's Send Pad Character should also be set to this value.

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

Timeout

0 – 99 [10]

For sending files with Kermit, specifies the number of seconds that the computer will wait after sending a packet before resending it.

BLASTscript variable: @KSTIMEOUT

For receiving files with Kermit, 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 filenames from local format to common Kermit format. Lower case is changed to all uppercase; and “~”, “#”, and all periods after the initial one are converted to “x”s.

NOTE: The reserved variable @KFNAMCONV is still supported and can be used to set both @KSNAMCONV and @KRNAMCONV to ON or

OFF. If @KSNAMCONV and @KRNAMCONV are set to the same values, a DISPLAY of @KFNAMCONV will show that value. If, however, @KSNAMCONV and @KRNAMCONV are set to the different values, a DISPLAY of @KFNAMCONV will return an error.

BLASTscript variable: @KSNAMCONV (files sent)

BLASTscript variable: @KRNAMCONV (files received)

Transfer Type **TEXT [BINARY]**

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

NOTE: The reserved variable @KFILETYP is still supported and can be used to set both @KSFILETYP and @KRFILETYP to ON or OFF. If @KSFILETYP and @KRFILETYP are set to the same values, a DISPLAY of @KFILETYP will show that value. If, however, @KSFILETYP and @KRFILETYP are set to the different values, a DISPLAY of @KFILETYP will return an error.

BLASTscript variable: @KSFILETYP (files sent)

BLASTscript variable: @KRFILETYP (files received)

Retry Limit **1 – 99 [10]**

Specifies the number of times that Kermit will attempt to send a single packet before aborting. For noisy connections, choose a higher setting.

BLASTscript variable: @KRETRY

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

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

Selecting **XMODEM**, **XMODEM1K**, **YMODEM**, or **YMODEM G** and pressing **ENTER** displays the subwindow shown in Figure 5-5. *Some fields apply to Xmodem only.*

FIGURE 5-5

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
Send Stripped Filename: NO	
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] LF

For Xmodem and Ymodem transfers, specifies how line termination is treated.

CR/LF – lines of text are terminated by a carriage return followed by a line feed (CR/LF); for example, when ASCII files are transferred to or from a DOS or Windows platform.

CR – lines of text are terminated by a carriage return (CR); for example, when ASCII files are transferred to or from a Macintosh platform.

LF – lines of text are terminated by a line feed (LF); for example, when ASCII files are transferred to or from a UNIX platform.

BLASTscript variable: @XYRLTS (files sent)

BLASTscript variable: @XYRLTR (files received)

Send Stripped Filename

YES [NO]

For sending files with Ymodem, specifies that the path and version number be stripped from the filename. This feature can prevent file transfer failures or incorporation of directory names into the remote filename due to system incompatibility.

BLASTscript variable: @YSTRIP

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 5-6 below.

FIGURE 5-6

ZMODEM Protocol	
Send Parameters	Receive Parameters
Resume interrupted file: NO	Auto Receive: YES
File must already exist: NO	File conversion: ASCII
Conversion override: ASCII	File management: CLOBBER
ASCII Line Termination: LF	Esc all control chars: NO
Send Stripped Filename: NO	
Management option: CLOBBER	
Esc all control chars: NO	
Limit block length: 0__	
Limit frame length: 0__	
Size of Tx window: 0__	
CRC: 32 BITS	

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

ASCII Line Termination [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 CR/LF.

BLASTscript variable: @ZMALT

Send Stripped Filename YES [NO]

For sending files with Zmodem, specifies that the path and version number be stripped from the filename. This feature can prevent file transfer failures or incorporation of directory names into the remote filename due to system incompatibility.

BLASTscript variable: @ZMSTRIP

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.

<u>Baud Rate</u>	<u>Block Length (in bytes)</u>
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 according to the file management option of the sender.

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

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

File Attributes Subwindow

Through the Default VMS File Attributes Setup submenu (Figure 5-7 below), the user can assign VMS file attributes for received files. To access this submenu, select the File Attributes setup field and press ENTER. The setup fields and settings for those setup fields are described below. For further details on the settings, consult VMS on-line Help or your VMS system user manual.

FIGURE 5-7

Default VMS File Attributes	
Binary File Attributes	Text File Attributes
File Organization: SEQ	File Organization: SEQ
Record Format: UDF	Record Format: UDF
Record Attributes: CR	Record Attributes: NONE
Control Area Size: 0	Control Area Size: 0
Record Size: 512	Record Size: 4096
Max Record Length: 0	Max Record Length: 0
Records Span Block: YES	Records Span Block: YES
Bucket Size: 0	Bucket Size: 0

File Organization

[SEQ] REL IDX

For binary and text files, specifies the RMS (Record Management System) file format. SEQ specifies Sequential, REL specifies Relative, and IDX specifies Indexed.

BLASTscript variable: @FILORGB (binary)

BLASTscript variable: @FILORGT (text)

Record Format

[UDF] FIX VAR VFC
STM STMLF STMCR

For binary and text files, specifies the RMS (Record Management System) record format. Possible settings are:

UDF – Undefined

FIX – Fixed

VAR – Variable

VFC – Variable length/fixed length control area

STM – Stream

STMLF – Stream/line feed

STMCR – Stream/carriage return

BLASTscript variable: @FILRFMB (binary)

BLASTscript variable: @FILRFMT (text)

Record Attributes

[NONE] FTN PRN **[CR]**

For binary and text files, specifies the RMS (Record Management System) record attributes. Possible settings are:

NONE – None

FTN – Fortran

PRN – Print

CR – Carriage return/carriage control

The default for binary files is NONE; the default for text files is CR.

BLASTscript variable: @FILRATB (binary)

BLASTscript variable: @FILRATT (text)

Control Area Size

[0] – 255

For binary and text files, specifies in bytes the size of the control area for VFC files.

BLASTscript variable: @FILFSZB (binary)

BLASTscript variable: @FILFSZT (text)

Record Size

0 – 32240 **[512]** **[4096]**

For binary and text files, specifies in bytes the size of the record. The default for binary files is 512; the default for text files is 4096.

BLASTscript variable: @FILLRLB (binary)

BLASTscript variable: @FILLRLT (text)

Max Record Length

[0] – 32767

For binary and text files, specifies in bytes the maximum record length.

BLASTscript variable: @FILMRSB (binary)

BLASTscript variable: @FILMRST (text)

Records Span Block

[YES] NO

For binary and text files, specifies whether a record can extend beyond a block boundary.

BLASTscript variable: @FILXBKB (binary)

BLASTscript variable: @FILXBKT (text)

Bucket Size

[0] – 63

For indexed binary and indexed text files, specifies the bucket size in 512-byte blocks.

BLASTscript variable: @FILBKSB (binary)

BLASTscript variable: @FILBKST (text)

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.

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, and Zmodem are examples of protocols widely used

by computer owners to transfer files. These file transfer protocols are fully described in the two 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 acknowledgement. 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 mini-computers.

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 a BLAST Session

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 VMS, 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 281), 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 setup fields of the BLAST Protocol Subwindow (see page 58).

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 on the next page).

FIGURE 6-1

```

BLAST Filetransfer default [000000USR]
Send Get Message Remote Local File
... send file(s) to the remote system
- local - 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 97 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 41 and the note concerning the Local menu and the Inactivity Timeout under the section “BLAST Protocol Timeouts” on page 83.

File – Executes a transfer command file that can control an entire filetransfer session unattended (see “Transfer Command File” on page 94). 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:

- ⌘ 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 ⌘ option.
- 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 165.

Sending a File

To send a file,

- ◇ First, select Send from the Filetransfer menu by pressing 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 with BLAST Protocol” on page 89). 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.

If you press ENTER without entering a remote filename, the path (if given) and filename entered at the local filename prompt will be specified as the remote filename. If you enter a filename, that filename (and path if given) will be specified as the remote filename. For an explanation of “%,” see “File Transfer Templates Using the ‘%’ Character” on page 88.

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 and then the local filename. 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.

NOTE: If you use wildcards, the target filenames will be in uppercase.

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) or version numbers.

IMPORTANT: “%” is REQUIRED for the target filename when the source filename contains a “%” or an “*” or when you want to strip the source path from the target filename when no target filename is given.

Some examples are:

<u>Source Name</u>	<u>Target Name</u>	<u>Result</u>
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 with BLAST Protocol

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.
- /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 62) 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. BLAST’s ability to set this switch is dependent on VMS permissions.
- /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 62) 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. BLAST’s ability to set this switch is dependent on VMS permissions.
- /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 126 and your system documentation for more information about permissions.

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 world
0002	Write by world
0001	Execute (search in directory) by world
0000	No permissions

System permissions will match owner settings, and delete permissions will match write settings. 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 62) 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
```

VMS-Specific Switches with BLAST Protocol

Using VMS-specific switches, you may specify certain VMS file attributes such as file organization, record attributes, and record format. All VMS file types are supported and transferred, but indexed files must be re-indexed on the destination computer. Indexed files may be transferred with all indexing information intact by backing them up into a saveset, transferring the saveset, and then restoring

them from the saveset. In most cases, this procedure will be faster than exporting the data to a flat file, transferring it, and re-indexing. The following table lists supported VMS-specific switches and their acceptable and default values.

VMS-Specific Switches with BLAST Protocol			
SWITCH	OPTIONS	DEFAULT	
		Binary	Text
/ORG= (File Organization)	VMSSEQ (sequential) VMSREL (relative) VMSINX (indexed)	VMSSEQ	VMSSEQ
/RTYPE= (Record Format)	UNDEF (undefined) VMSFIXED (fixed) VMSSTRM (stream) VMSCR (stream/carriage return) VMSLF (stream/line feed) VMSVAR (variable) VMSVFC (variable length with fixed control area size)	UNDEF	VMSVAR
/RATTR= (Record Attribute)	NONE (none) VMSFORT (Fortran) VMSPRNT (print) VMSRETN (carriage return/carriage control)	NONE	VMSRETN
/VFCSIZE= (Fixed Control Area Size)	<i>nnn</i> (0 – 255 bytes)	0	0
/LRECL= (Record Size)	<i>nnn</i> (0 – 32240 bytes)	512	4096
/MAXREC= (Maximum Record Length)	<i>nnn</i> (0 – 32767 bytes)	0	0
/XBLK= (Records Allowed to Cross Block Boundaries)	YES NO	YES	YES
/BKSIZE= (Bucket Size)	<i>nnn</i> (0 – 63)	0	0

For example, to send a text file named DATA.TXT, the following could be used for the file specifier:

```
DATA.TXT/ORG=VMSSEQ/RTYPE=VMSLF/RATTR=CR/LRECL=512
```

Filename Restrictions with BLAST Protocol

If you are transferring files to or from another system using forward slashes as directory delimiters, 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) as well as the above mentioned VMS switches.

You can work around this restriction by changing your remote working directory to the one into which you want to transfer the file and giving the filename without a path. To change your remote working directory interactively, choose the Chdir command from the Remote menu. Alternatively, you may do a scripting workaround:

```
filetransfer
  remote
    chdir                # Change working directory
      [USER.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
  send
  APP                    # Filename only--no path
  [USER.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 *ATTNATTN*. 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: This feature is not supported for text files or for files with the /STR switch.

Transfer Command File

A transfer command file (TCF) 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 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 message, which will be transmitted to the remote display and the remote log, for example:

;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 the command to be sent to the remote computer, for example:

```
!dir
```

The valid remote 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.
- ◇ The file 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 VMS user can delete a file on a UNIX remote system without actually typing the UNIX 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 Ccmds setup field (page 62) 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 of the 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, "JOE*.TXT"), 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, “JOE*.TXT”).

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 Protocol” on page 165.

Fine-Tuning the BLAST Session Protocol

Packet Size

Most computers can process packets of 256 characters. Set the Packet Size setup field (page 64) 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 set-up fields (pages 62 – 63). 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.

Disabling File Overwrites and Remote Commands

The Enable /OVW and Remote Cmds setup field (page 62) and the script variable @ENABLERCMD (page 240) 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 62) and the @ENABLEFS (page 240) 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 with BLAST Protocol” on page 89.

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 runs a specific BLAST setup when a user logs into your system, 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 61) or by setting the @TRPASSWD reserved variable (page 264) in a slave script.

NOTE: The transfer password is superseded by the Secure BLAST password (see “Using Secure BLAST” on page 127).

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 165 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

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 7-1 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.

```

BLAST Kermit default [000000.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 >>-----

```

FIGURE 7-1

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 109).

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 168.

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 88), 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 88). 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 106). 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 67) 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 67) 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:

- | | |
|-------------------------------|---|
| <code>/APP</code> | Append to a file with the same name if it exists. |
| <code>/GROUP=<i>nn</i></code> | Preserve or set the group of the file where <i>nn</i> is a positive decimal integer that specifies the file group ID. BLAST’s ability to set this switch is dependent on VMS permissions. |

- `/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. BLAST's ability to set this switch is dependent on VMS permissions.
- `/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 126 and your system documentation for more information about permissions.

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 world
0002	Write by world
0001	Execute (search in directory) by world
0000	No permissions

System permissions will match owner settings, and delete permissions will match write settings. 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.

VMS-Specific Switches with Kermit

Using VMS-specific switches, you may specify for files received certain VMS file parameters such as file organization, record attributes, and record format. All VMS file types are supported and transferred, but indexed files must be re-indexed on the destination computer. Indexed files may be transferred with all indexing information intact by backing them up into a saveset, transferring the saveset, and then restoring them from the saveset. In most cases, this procedure will be faster than exporting the data to a flat file, transferring it, and re-indexing. The following table lists supported VMS-specific switches and their acceptable and default values. For example, to send a text file named DATA.TXT, the following could be used for the file specifier:

VMS-Specific Switches Using Kermit Protocol			
SWITCH	OPTIONS	DEFAULT	
		Binary	Text
/ORG= (File Organization)	VMSSEQ (sequential) VMSREL (relative) VMSINX (indexed)	VMSSEQ	VMSSEQ
/RTYPE= (Record Format)	UNDEF (undefined) VMSFIXED (fixed) VMSSTRM (stream) VMSCR (stream/carriage return) VMSLF (stream/line feed) VMSVAR (variable) VMSVFC (variable length with fixed control area size)	UNDEF	VMSVAR
/RATTR= (Record Attribute)	NONE (none) VMSFORT (Fortran) VMSPRNT (print) VMSRETN (carriage return/car- riage control)	NONE	VMSRETN
/VFCSIZE= (Fixed Control Area Size)	<i>nnn</i> (0 – 255 bytes)	0	0
/LRECL= (Record Size)	<i>nnn</i> (0 – 32240 bytes)	512	4096
/MAXREC= (Maximum Record Length)	<i>nnn</i> (0 – 32767 bytes)	0	0
/XBLK= (Records Allowed to Cross Block Boundaries)	YES NO	YES	YES
/BKSIZE= (Bucket Size)	<i>nnn</i> (0 – 63)	0	0

DATA .TXT/ORG=VMSSEQ/RTYPE=VMSLF/RATTR=CR/LRECL=512

See “Sending and Receiving Files with Kermit” on page 104 for more on scripting for Kermit.

Kermit Remote Menu

Notice that the Kermit Remote menu (Figure 7-2 below) 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.

```
BLAST KermitRemote default [000000.USR] MENU
Directory Erase Type Cwd Space Who Message host Kermit Help
... list remote filenames
-- local ----- % xfer - file size - byte cnt - retries -
S: <idle>
R: <idle>
----- ?-help - ESC-exit -----

-----<< Entering KERMIT Transfer Mode >>-----
```

FIGURE 7-2

The Remote menu commands are:

Directory – Displays the server’s current working directory or a directory you specify; wildcards can be used.

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 8

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 either near-transparent remote access to other operating systems or 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:

```
blast -enumber
```

where *timeout* is equal to *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.

You can also select the pad character for Xmodem using the following format:

```
blast -px
```

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” on page 176 for details.

Invoking a command line parameter affects these protocols only for the duration of that communications session.

Xmodem Protocol

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. You may change your er-

ror-detection setting through the Error Detection setup field (page 68) 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 171.

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.

Optionally, you may add the /TXT file transfer switch (see “File Transfer Switches with Xmodem” on page 113).

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, it will automatically be overwritten.

Optionally, you may add any supported file transfer switches (see the next section, “File Transfer Switches with Xmodem”). For example, you may append to an existing file by adding the /APP switch to the local filename when prompted for the name.

File Transfer Switches with Xmodem

Xmodem supports several file transfer switches; it ignores all switches that it does not support.

File Transfer Switches with Xmodem		
<code>/APP</code>	Receive	Append to a file with same name if it exists.
<code>/FWD</code>	Send	Delete file from sending system if the transfer was successful. NOTE: The <code>/FWD</code> switch is a very powerful feature. Because it allows files to be automatically <i>deleted</i> from the sending system, always exercise caution when using it.
<code>/GROUP=<i>nn</i></code>	Receive	BLAST's ability to set this switch is dependent on VMS permissions. Preserve or set the group of the file where <i>nn</i> is a positive decimal integer that specifies the file group ID. BLAST's ability to set this switch is dependent on VMS permissions.
<code>/OVW</code>	Receive	Overwrite a file with the same name if it exists.
<code>/OWNER=<i>nn</i></code>	Receive	BLAST's ability to set this switch is dependent on VMS permissions. Reserve or set the owner of the file, where <i>nn</i> is a positive decimal integer that specifies the file owner ID. BLAST's ability to set this switch is dependent on VMS permissions.
<code>/PERMS=<i>nnnn</i></code>	Receive	Preserve or set file permissions where <i>nnnn</i> 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 126 and your system documentation for more information about permissions 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 world 0002 Write by world 0001 Execute (search in directory) by world 0000 No permissions

		System permissions will match owner settings, and delete permissions will match write settings. 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.

VMS-Specific Switches with Xmodem

Using VMS-specific switches, you may specify for files received certain VMS file parameters such as file organization, record attributes, and record format. All VMS file types are supported and transferred, but indexed files must be re-indexed on the destination computer. Indexed files may be transferred with all indexing information intact by backing them up into a saveset, transferring the saveset, and then restoring them from the saveset. In most cases, this procedure will be faster than exporting the data to a flat file, transferring it, and re-indexing. The following table lists supported VMS-specific switches and their acceptable and default values.

VMS-Specific Switches with Xmodem			
SWITCH	OPTIONS	DEFAULT	
		Binary	Text
/ORG= (File Organization)	VMSSEQ (sequential) VMSREL (relative) VMSINX (indexed)	VMSSEQ	VMSSEQ
/RTYPE= (Record Format)	UNDEF (undefined) VMSFIXED (fixed) VMSSTRM (stream) VMSCR (stream/carriage return) VMSLF (stream/line feed) VMSVAR (variable) VMSVFC (variable length with fixed control area size)	UNDEF	VMSVAR

SWITCH	OPTIONS	DEFAULT	
		Binary	Text
/RATTR= (Record Attribute)	NONE (none) VMSFORT (Fortran) VMSPRINT (print) VMSRETN (carriage return/carriage control)	NONE	VMSRETN
/VFCSIZE= (Fixed Control Area Size)	<i>nnn</i> (0 – 255 bytes)	0	0
/LRECL= (Record Size)	<i>nnn</i> (0 – 32240 bytes)	512	4096
/MAXREC= (Maximum Record Length)	<i>nnn</i> (0 – 32767 bytes)	0	0
/XBLK= (Records Allowed to Cross Block Boundaries)	YES NO	YES	YES
/BKSIZE= (Bucket Size)	<i>nnn</i> (0 – 63)	0	0

For example, to send a text file named DATA.TXT, the following could be used for the file specifier:

```
DATA . TXT /ORG=VMSSEQ /RTYPE=VMSLF /RATTR=CR /LRECL=512
```

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 172.

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 88).

Optionally, you may add the /TXT and /FWD file transfer switches. You will not be able to add any other file transfer switches (see “File Transfer Switch Using 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 with Ymodem

Ymodem does not support switches on receiving filenames; on sending filenames, it supports the the following two switches:

File Transfer Switches with Ymodem		
/FWD	Send	Delete file from sending system if the transfer was successful. NOTE: The /FWD switch is a very powerful feature. Because it allows files to be automatically <i>deleted</i> from the sending system, always exercise caution when using it.
/TXT	Send	Send file as ASCII with value stored in @XYRLTS (page 268). NOTE: Ymodem text translation should only be used on files whose Record Format is VARIABLE. Text translation on files with other Record Formats may give unexpected results.

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 69).

The following two sections describe interactive file transfers. For a discussion of scripting Zmodem file transfers, see “File Transfers with Zmodem” on page 174.

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 88).

Optionally, you may add supported file transfer switches (see “File Transfer Switches Using 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 with Zmodem

Zmodem supports several file transfer switches for sending file-names (see table on the next page). Zmodem cannot set switches on receiving filenames and ignores all unsupported switches.

File Transfer Switches with Zmodem		
/APP	Send	Specify APPEND as File Management option.
/FWD	Send	Delete file from sending system if the transfer was successful. NOTE: The /FWD switch is a very powerful feature. Because it allows files to be automatically <i>deleted</i> from the sending system, always exercise caution when using it.
/OVW	Send	Specify CLOBBER as File Management option.
/TXT	Send	Send file as ASCII with value stored in @ZMALT.

Chapter 9

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 VMS system, for example, you might type:

```
edit remote.fil
```

which instructs `edit` to open a new file named `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 `ATTNATTN` 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 `SYSTEMS.SCR` library to automate the process. BLAST will start the remote text capture program for you.

BLASTscript Automatic Method

See “Text Transfers” on page 178 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.
- ◇ 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 178 for details on scripting downloads.

Chapter 10

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 VMS methods of security.

IMPORTANT: There are many tools for securing a VMS 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 VMS security.

VMS Tools

To maintain security, each person logging into your VMS 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 a group. Segregating users into groups can help secure your system. For more information on setting up groups, refer to your system documentation.

Permissions

The basic operations performed on a file are “read,” “write,” “delete,” and, for executable files, “execute.” Permissions, or file protection, can be set to grant or deny access to a file for any of these operations. Read, write, delete, and execute file protection can be set for the system, for the owner of the file, for users in a particular group, and for all other users (referred to as “world”) on the system.

When you do a full listing for a file or issue a **show security** command for a file, the file protection assigned to the file appears as a series of letters. “Read,” “write,” “delete,” and “execute” are denoted by the mnemonics “r,” “w,” “d,” and “e.” For example, a file might have the following permissions:

```
Protection: (System: RWED, Owner: RWED, Group: RE, World: R)
```

The system and owner of the file have all four permissions, users in the same group as the owner have only read and execute permission, and all other users have only read permission.

As a general rule, set default permissions for newly created files as restrictively as possible. The command for setting default permissions is:

```
set protection=(s:xxxx,o:xxxx,g:xxxx,w:xxxx)/default
```

where **s**, **o**, **g**, **w** stand for system, owner, group, and world, respectively, and **xxxx** is the file protection designation (any combination of RWED). If the owner of a file wants to allow expanded access to his files later, he can manually reset the file protection designation by using the **set security** command. For more information on changing the file protection designation for a file, see your system documentation.

Directories

To the VMS operating system, a directory is just a file. The same read, write, delete, 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, delete, and execute permission for the owner only. This file protection setting will allow the owner of the directory complete access to his or her files but disallow access by all others.

BLAST Protocol File Transfer and Permissions

When a file is transmitted using BLAST protocol, permissions are set according to the following rules (see page 90 for information on the /PERMS switch and other switches that affect permissions):

- ◇ If the file is being transmitted from a VMS 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 VMS permission structure, permissions will be set according to the closest equivalent on the receiving system.

Using Secure BLAST

Secure BLAST was developed to provide an extra layer of security beyond existing VMS restrictions. In addition to recognizing VMS file protection settings, Secure BLAST can further restrict access to particular files and insure that a user executes only authorized versions of BLAST on both the local and remote systems. Furthermore, Secure BLAST can limit authorized users to a 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 140.

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 a 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

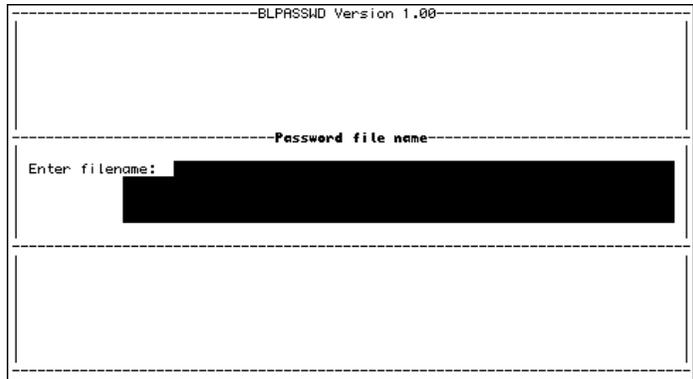
The first step in securing BLAST is to create the password file. The application BLPASSWD provides a full-screen user interface for setup and maintenance of the password database. The BLAST installation program normally copies BLPASSWD to the same directory as the BLAST executable. For increased security, 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

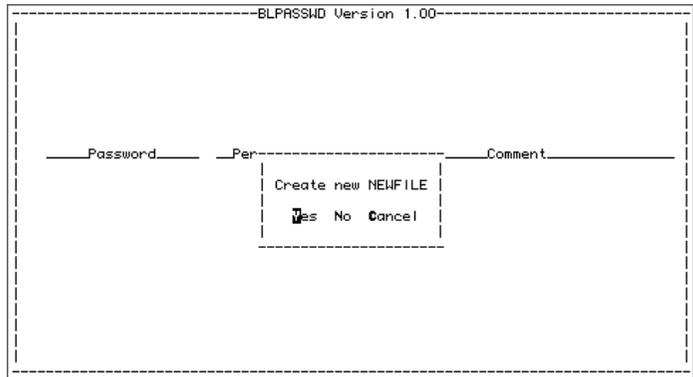
be prompted for a filename (Figure 10-1). Type the filename and press ENTER.

FIGURE 10-1



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 10-2). Press Y to create the file, or press N or C to cancel and exit BLPASSWD.

FIGURE 10-2



After creating a new password file, BLPASSWD will display the main screen (Figure 10-3, next page). 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. 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.

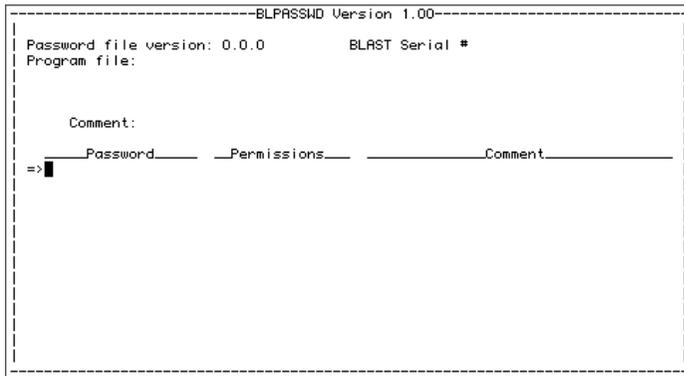


FIGURE 10-3

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 for 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 10-4 (next page). Type the appropriate information into the field highlighted and then move to the next field by pressing **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.*

```

-----BLPASSWD Version 1.00-----
----- Modify Header -----
Password: ██████████
Serial Number:
BLAST filename:

Comment:

-----

```

FIGURE 10-4

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 XXXXXXXXXXXX-X-XXXXXX

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. To display the serial number and version of BLAST while running BLAST, press the *HELP* key. *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, [USER . JOE] BLAST . EXE where [USER . JOE] is the directory location and BLAST . EXE is the name of the secured executable.

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 (deactivate) a record and its password. The record will be marked as unused but not physically removed. When a record is zapped, a “z” is displayed after the permissions.

A zapped password cannot be used for a new record until the password has 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 reactivated.
- R Reclaim zapped password for possible reuse and delete zapped record. Record numbers may change after use of this command.
- Q Quit BLPASSWD.

After designating a file for creation or editing using the command keys, press ENTER; BLPASSWD will display a screen similar to the one shown in Figure 10-5 below. Type the appropriate information into the highlighted field and then move to the next field by pressing 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.

```

-----BLPASSWD Version 1.00-----
|----- Add New Record -----|
|                               |
| Password: ██████████         |
| Permissions: (GSTLERPCOA or M)|
| Serial Number:                |
| Home Directory:                |
|                               |
| Include mask:                  |
|                               |
| Exclude mask:                  |
|                               |
| Comment:                       |
|                               |
|-----|
  
```

FIGURE 10-5

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 on next page).

IMPORTANT: BLAST does not override standard VMS file protection settings. For example, even though a user may have BLAST permission to rename a file, he cannot do so if he does not have VMS write permission for that file. Likewise, he cannot change directories if he does not have VMS 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 XXXXXXXXXXX-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. To display the serial number and version of BLAST while running BLAST, press the *HELP* key.

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 VMS 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 change-directory permission and VMS 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 the specified 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 VMS read, write, delete, and execute file protection is 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 VMS read, write, delete, 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 an optional comment regarding each record.

BLSECURE

The application `BLSECURE` is a command line utility that, like `BLPASSWORD`, sets up and maintains passwords and permissions for BLAST users. Unlike `BLPASSWORD`, 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, BLSECURE 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 cur-

rently 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 10-6 below:

FIGURE 10-6

test3	(master password)
10.7.7	(password file version)
0123456789-0-00000	(host BLAST serial number)
newfile	(password filename)
[USER.JOE]BLAST.EXE	(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). The entry and password must be new. See “BLPASSWD” on page 128 for complete descriptions of the data fields associated with the following arguments.

<i>pwd</i>	Password for the record.
<i>perm</i>	Permission specifier as described below.
<i>sn</i>	Serial number of the remote BLAST executable.
<i>home</i>	Directory to which the host computer changes upon linking in Filetransfer mode.
<i>inc</i>	Include mask for specifying files that may be obtained by the remote user.
<i>exc</i>	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.
<i>comment</i>	Optional comment.

The following are the possible hexadecimal 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 10-6 (preceding page) and 10-7 below, you would type the following:

```
blsecure newfile test3 a site23 0001 1234567891-0-00000 -
[user.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 10-7 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.

FIGURE 10-7

site23	(password)
0001	(permission in hexadecimal)
1234567891-0-00000	(remote serial number)
[USER.SITES]	(home directory on host)
*23.DAT	(include file mask)
MAST*.*	(exclude file mask)
Password for site 23	(comment)

f *pwd*

searches for a record by its password and displays the record as shown in Figure 10–7 (preceding page).

pwd Password for the individual record.

z *pwd*

zaps (deactivates) a record by marking it for deletion but not physically removing it. When the zap command is used on an already zapped record, the record is unzapped and the password and record are reactivated. 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 re-ordered by using *r*.

BLSECURE Error Codes

Errors that can occur while running BLSECURE are usually 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 [user.joe]blast.exe -s [private]password.fil
```

In this example, `[user.joe]blast.exe` is the pathname for the BLAST executable, 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 [user.joe]blast.exe -d
```

will respond with a message similar to the following:

```
Secure - Version 1.0  
[private]password.fil ([user.joe]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 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 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 via the Send command of the Filetransfer menu or with a BLASTscript FILETRANSFER SEND statement. If a Send menu 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=test3

  send
  UPDATE23.DAT
  SITE23.DAT
  ta
esc
```

Since no remote filename or transfer options are used, two blank lines follow the password line. See “File Transfers with BLAST Protocol” on page 165 for information on scripting file transfers.

Chapter 11

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 completion of the currently executing statement, 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 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 the
[ 4] # insurance company
[ 5] #
[ 6] # Section 1: CONNECTING
[ 7] #
[ 8] .BEGIN
[ 9] set @ONERROR = "CONTINUE"
[10] connect
[11] if OK 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 eq "0"
[25]     display "An error occurred during 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 233). @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 181).

Line 11 demonstrates how scripts are programmed to make choices with the IF (conditional) statement. After the CONNECT command executes, @STATUS is set to indicate whether or not the connection was successful. The IF statement tests the success of the CONNECT command. If the connection was successful, the script performs the GOTO command, sending the script to the section labeled .XFER, which controls file transfer.

```
if OK goto .XFER
```

*conditional executes if conditional
statement clause is true*

If the CONNECT command is not successful, the 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 49).

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 infor-

mation on scripting for alternative protocols, see Chapter 12). 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: If Emulation is set to `PASSTHRU`, Learn mode will not record Terminal mode interactions.

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 you 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 NOT OK 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 12, which describes techniques for working with disk files, manipulating strings, and interacting with programs in your system. Chapter 13 discusses

the BLAST method of connecting and disconnecting, which relies heavily on scripts. Chapters 14 and 15 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 30).

Chapter 12

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 or a `FILETRANSFER-ESC` 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 eq "0" set @mystat = "GO"
```

```
if @STATUS eq "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

```
if @STATUS eq "0"

disconnect

end
return
```

LEGAL

```
if @STATUS eq "0"
#
disconnect
#
end
return
```

A typing mistake in a script line can also generate an error message. For example, a line such as

```
ig @STATUS eq "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 eq "0" display "Disconnected OK"
# else display "Disconnect failure!"
# if @fruit eq "0" display "No fruit was selected"
# if @fruit eq "1" display "Apples are delicious"
# if @fruit eq "2" display "Oranges are tasty"
# if @fruit eq "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 193.

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 is ", @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 eq "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...
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.

`STRRIX 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 14.

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 + "3"
  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 CARRIER`, `WAIT IDLE`, `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 char-

acters 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 14.

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 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 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.
#
clear
set @file = "MODEMS.SCR"
```

```

fopenr 1, @file
if NOT OK
    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 line read, count line and return for another
#
if OK
    let @count = @count + "1"
    cursor 10, 21
    put @count
    goto .LOOP
end
fwrite 2, @count, " lines in MODEMS.SCR."
fclose 1
fclose 2
display "Done! Check LINE.CNT for line count."
return
# End of script.

```

Managing the Screen Display

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.

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 198. 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
#
set @ONERROR = "CONTINUE"
set @USERIF = "OFF"
clear                # Erase the screen
cursor 12, 30
put "Now connecting, please wait."
set @SCRLREG = "OFF"
connect
if NOT OK
    set @SCRLREG = "ON"
    set @USERIF = "ON"
    clear
    write "Can't connect or log in."
    return
end
set @SCRLREG = "ON"
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
set @syscmd = "cp MODEMS.SCR MODEMS.TXT"
local
    system
    @syscmd
esc
return
# End of script.
```

File Transfers with BLAST Protocol

Chapter 6 describes the BLAST 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 148) 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]   t a
[ 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. Transfer options may also be indicated by adding file transfer switches to filenames (see “File Transfer Switches with BLAST Protocol” on page 89).

In the SEND example, two blank lines (lines 8 and 9) are entered to indicate that the remote filename will be the same as the local filename and that no transfer options will be used (thus, the file transfer will be binary).

Blank lines representing default filenames and file transfer options (`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
      [USER.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 94). 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 41). 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 GETs) before issuing local commands (like SENDs). 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 177.

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 terminal device. If BLAST does not react to changes in DCD as expected, consult your system documentation.

File Transfers with Kermit

Before writing scripts for Kermit, you may want to review the general information in Chapter 7, *Kermit Protocol*, on page 103. Learn mode (page 148) is also a good tool for obtaining a rough draft of a script you will need in a particular case. For a list of file transfer switches supported by Kermit, see “File Transfer Switches with Kermit” on page 106.

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
        [remote_filename]
esc
```

NOTE: You must specify a remote filename in either the receive command or the FILETRANSFER-ESC block. If you specify a local filename in both places, the filename in the receive command will be used, and the filename in the FILETRANSFER-ESC block will be ignored.

Kermit Server

Before issuing a SEND command, you must start Kermit server on the remote machine. For most VMS machines, this command is "server." The basic syntax for sending files using Kermit server is as follows:

```
connect
tsend "kermit", CR
tsend "server", CR
filetransfer
    send
        local_filename
        remote_filename
esc
```

Receiving Files

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[file_transfer_switch(es)]
esc
```

Kermit Server

Before issuing a GET command, you must start Kermit server on the remote machine. For most VMS machines, the command is "server." The basic syntax for GETs using Kermit server is as follows:

```
connect
tsend "kermit", CR
tsend "server", CR
filetransfer
    get
        remote_filename
        local_filename[file_transfer_switch(es)]
esc
```

Transferring More Than One File

Unless you exit Kermit 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", CR
tsend "server", CR
filetransfer
    get
        SALESREPORT.TXT
        STORE1SALES.TXT/OVW
    send
        STORE1INVENTORY.TXT
        INVENTORY.TXT
    finish
esc
```

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/OVW
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 177.

File Transfers with Xmodem and Xmodem1K

Before writing scripts for Xmodem and Xmodem1K, you may want to review the general information in Chapter 8 on the use of these protocols. Learn mode (page 148) is also a good tool for obtaining a rough draft of the script you will need in a particular case. For a list of file transfer switches supported by Xmodem and Xmodem1K, see “File Transfer Switches with Xmodem” on page 113.

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[file_transfer_switch(es)]
esc
```

Receiving Files

The syntax for receiving files is:

```
connect
tsend "send_command remote_filename", CR
filetransfer
  get
  local_filename[file_transfer_switch(es)]
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.TXT", CR
filetransfer
  send
  S1SALES.TXT
esc
tsend "rx ORDER.TXT", CR
filetransfer
  send
  S1ORDER.TXT
esc
tsend "sx INVENTORY.TXT", CR
filetransfer
  get
  S1INVENTORY.TXT/OVW
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 177.

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 8 on the use of these protocols. Learn mode (page 148) 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. For a list of file transfer switches supported by Ymodem and Ymodem G, see “File Transfer Switches with Ymodem” on page 117.

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[file_transfer_switch(es)]
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.TXT
esc
tsend "rb", CR
filetransfer
    send
```

```
ORDER.TXT
esc
tsend "sb 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 177.

File Transfers with Zmodem

Before writing scripts for Zmodem, you may want to review the general information in Chapter 8. Learn mode (page 148) is also a good tool for obtaining a rough draft of a script. For a list of file transfer switches supported by Zmodem, see “File Transfer Switches with Zmodem” on page 119.

The Zmodem protocol is configured through the Zmodem setup sub-window. 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[file_transfer_switch(es)]
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 "rz", CR
filetransfer
    send
        ORDER.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 177.

BLAST Operation as a Pseudohost

If a remote user logs onto your VMS 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. 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 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 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 accidentally 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 OK                             # successful read
    strinx @logline, "send complete" # crucial!
    if @STATUS eq "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
```

Text Transfers

The following section describes scripting for text transfers. See *Text Transfers* on page 121 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 VMS using the text editor `lse`. The script TTRAPS for the filename used in `lse`'s exit status line; the WAIT command gives the buffers on the local and remote computers time to clear.

```
connect
tsend "lse CIH4.TXT", CR # Send cmd to start remote editor
wait 3
tsend "^z" # Put lse in cmd mode
tsend "goto bottom", CR # Move cursor to end of file
```

```

tsend "^z"                # Take lse out of cmd mode
tupload "CIH4.TXT"
wait 3 idle
tsend "^z"                # Put lse in command mode
wait 1
tsend "exit", CR          # Exit lse
ttrap 30, "CIH4.TXT"      # trap filename-exit status line
set @hold = @status
wait 3 idle
if @hold eq "0"
    display "Tupload not completed."
    return
end
else display "Tupload successful."
wait 10

```

For more specific error checking, check @STATUS for TUPLOAD:

```

connect
tsend "lse CIH4.TXT", CR # Send cmd to start lse
wait 3
tsend "^z"                # Put lse in command mode
tsend "goto bottom", CR  # Move cursor to end of file
tsend "^z"                # Take lse out of cmd mode
tupload "CIH4.TXT"
set @hold1 = @status
wait 3 idle
if @hold1 eq "0" display "Tupload cmd execution complete."
else
    display "Tupload cmd failure; error ", @hold1
    tsend "^z"            # Put lse in command mode
    wait 3
    tsend "quit", CR      # Quit lse without saving file
    return
end
tsend "^z"                # Put lse in command mode
wait 1
tsend "exit", CR          # Exit lse
ttrap 30, "CIH4.TXT"
set @hold2 = @status
wait 3 idle
if @hold2 eq "0"
    display "Tupload not completed."
    return
end
else display "Tupload completed."

```

Downloading Text

To download a text file from within a script, write a BLAST script that includes a TCAPTURE statement, which 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 "type 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 13

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 mo-

dem 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 in 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 in 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 defined by the BLASTDIR symbol 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. 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. If the Modem Type setup field is empty, MODEMS.SCR is not called.

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, the system script is not called.

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 turning it off and then turning it on again. 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 186). 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 eq "-1" return 1
  end
  tsend "ATDT", @PHONENO, CR
  ttrap 45, "CONNECT", "NO CARRIER", "BUSY", "NO DIAL"
  if @STATUS eq "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.

The Index Utility

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 14

BLASTscript Command Reference

Introduction

As you learned in Chapter 11, 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 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 numeric constants or numeric strings or variables containing numeric constants or numeric strings.

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:

<code>\r</code>	carriage return
<code>\n</code>	linefeed
<code>\f</code>	formfeed
<code>\b</code>	backspace
<code>\t</code>	tab
<code>\\</code>	backslash character
<code>'</code>	quotation marks
<code>\xxx</code>	where xxx is the three-digit octal value of the character except for the octal value of null (<code>\000</code>), 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 15 for more information.

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 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:

ASCTII	FOPENW	LOCAL SYSTEM	STRINX
ASK	FREAD	LPRINT	STRLEN
CALL	FREADB	LRENAME	SYMTYPE
CONNECT	FREWIND	LTYPE	TCAPTURE
DISCONNECT	FWRITE	NEW	TSEND
DROP	FWRITEB	RAISE	TSENBIN
FCLOSE	LCHDIR	REMOVE	TTRAP
FILETRANSFER	LDELETE	RETURN	TUPLOAD
FOPENA	LLIST	SELECT	WAIT CARRIER
FOPENR	LOAD	STRINX	WAIT IDLE

Additionally, the following commands now set @STATUS *when* the commands assign a value to a reserved variable associated with a setup field: ASK, FREAD, LET, LOWER, SET, STRTRIM, and UPPER. If the assignment of the value is successful, @STATUS is set to 0; if the assigned value is invalid, @STATUS is set to 1. If the command is not completed for some other reason, @STATUS is set to a nonzero value. For example, if in responding to an ASK statement the user presses ESC instead of ENTER, @STATUS is set to -1.

IMPORTANT: Because this behavior makes debugging scripts more difficult, we discourage using setup-field reserved variables with the following commands: ASK, FREAD, LET, LOWER, STRTRIM, and UPPER.

Manipulation of Binary Data

BLAST Professional VMS permits manipulation of binary data using the reserved variables @FILECNT, @SYMBOLTYPE, and @TRAPCNT (see Chapter 15) and the following BLASTscript commands—BIN2HEX, CHECKSUM, FREADB, FWRITEB, HEX2BIN, SYMTYPE, TRAPNULLS_ON, and TSENBIN. For a full discussion of the function of these commands, see the following description of specific commands.

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.

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 eq "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 “Commands That Set @STATUS” on page 193).

EXAMPLE:

```
ask "what month", @month
ask noecho "Password?", @secret      # no display
```

BIN2HEX

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. If the called script does not exist, @STATUS is set to 1; if the script is successfully called, @STATUS is set to 0.

On return from the called script, @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 eq "0" display "Backup Successful"
```

CHECKSUM

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; *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 MODEMS.SCR and SYSTEMS.SCR libraries to dial the modem and log on if the Modem and System Type setup fields are specified. If CONNECT is successful, @STATUS is set to 0. For more information about the operation of the CONNECT command, see Chapter 13.

EXAMPLE:

```
connect
if OK 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 to 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 the SYSTEMS.SCR and MODEMS.SCR libraries to log off and hang up the modem if the System and Modem Type setup fields are specified. If DISCONNECT is successful, @STATUS is set to 0. See Chapter 13 for a full discussion.

EXAMPLE:

```
disconnect
if OK 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 terminal.

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 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 NOT OK
  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
fwrite 1, "No update today."    #
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 94 for a complete description of the transfer command file format.

EXAMPLE:

```
filetransfer
  file
  COMMAND.TCF
esc
disconnect
quit
```

FILETRANSFER GET / SEND

get/send file

<i>FORMAT:</i>	<i>FILETRANSFER</i>	<i>FILETRANSFER</i>
	<i>GET</i>	<i>SEND</i>
	<i>{protocol-dependent string(s) ...}</i>	<i>{protocol-dependent string(s) ...}</i>
	<i>ESC</i>	<i>ESC</i>

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 Protocol” on page 165 and the sections on scripting file transfers for the other supported protocols in Chapter 12.

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
```

FILETRANSFER LOCAL

perform local commands using BLAST protocol

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. 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 defined by the BPRINTER symbol. The line following PRINT is the filename (see LPRINT on page 217).

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
    DATA.DAT
    rename
    DATA.DAT
    DATA.BAK
  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 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.

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	

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.

IMPORTANT: FOPENA treats files in the same manner as the VMS system TYPE command except for files with Record Format set to VFC. Files with Record Format set to VFC will not be opened properly using FOPENA.

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.

IMPORTANT: FOPENR treats files in the same manner as the VMS system TYPE command except for files with Record Format set to VFC. Files with Record Format set to VFC will not be opened properly using FOPENR.

EXAMPLE:

```
fopenr 1, "COMMAND.TCF"         # 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 is 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.

IMPORTANT: FOPENW treats files in the same manner as the VMS system TYPE command except for files with Record Format set to VFC. Files with Record Format set to VFC will not be opened properly using FOPENW.

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. The reserved variable @FILECNT stores the actual number of bytes read.

IMPORTANT: FOPENA, FOPENR, and FOPENW treat files in the same manner as the VMS system TYPE command except for files with Record Format set to VFC. Files with Record Format set to VFC will not be opened properly using FOPENA, FOPENR, or FOPENW. When using FREAD, keep in mind this limitation; also keep in mind how the VMS system TYPE command opens files.

EXAMPLE:

```
fopenr 1, "COMMAND.TCF"     # open file 1 for reading
fread 1, @input             # read line into @input
if NOT OK display "End of file reached"
fclose 1                    # close file
```

FREADB

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. The reserved variable @FILECNT stores the actual number of bytes written. If FWRITE is successful, @STATUS is set to 0.

IMPORTANT: FOPENA, FOPENR, and FOPENW treat files in the same manner as the VMS system TYPE command except for files with Record Format set to VFC. Files with Record Format set to VFC will not be opened properly using FOPENA, FOPENR, or FOPENW. When using FWRITE, keep in mind this limitation; also keep in mind how the VMS system TYPE command opens files.

EXAMPLE:

```
fopenw 1, "OUTPUT.FIL"  
fwrite 1, "the userid is: ", @USERID  
fclose 1
```

FWRITEB

write a file as binary data

FORMAT: *FWRITEB numeric_value1, string_value, numeric_value2*

FWRITEB writes up to a maximum number of bytes (*numeric_value2*) from *string_value* 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. If FWRITEB is successful, @STATUS is set to 0.

EXAMPLE:

```
fopena 2, "SALES.TXT"  
fwriteb 2, @line, 100 # writes up to 100 bytes from @line  
# into file 2.
```

GETENV

store the value of a symbol

FORMAT: *GETENV string_value, variable*
 GETENV writes the value of a symbol (*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 script specified by *.LABEL*, which can be up to eight characters (not counting the initial period) and is case-insensitive. If *.LABEL* cannot be found, the script is aborted.

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

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. A character by character comparison of the two strings is made. The comparison stops when a difference in ASCII values is encountered. The string containing the character with the higher ASCII value is considered to be greater. If the first string difference encountered is the end of one of the strings, the longer string is considered to be greater.

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 157 for more information on using REPS and loops).

[NOT]EXIST *string_value*
True [False] when a file named the value of *string_value* exists.

[NOT]ISDIR *string_value*
True [False] when *string_value* is a directory.

[NOT]OK
True [False] when @STATUS equals 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 eq "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 OK 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 OK                            # if cmd successful
  display "chdir ok"
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 OK 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 some accompanying data rather than just the filenames and directory names. @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 defined by the SETUPDIR symbol. *String_value* is the name of the setup. If the setup is in a subdirectory of the directory defined 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 OK display "Setup Blaster is the current setup"
else display "can't load the setup Blaster"
```

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 `logout` 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 = "dir /full"
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 defined by the *BPRINTER* symbol (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 OK 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 OK 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 LTYPE is successful, @STATUS is set to 0.

EXAMPLE:

```
ltype "salesdata" # display salesdata
```

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 defined by the SETUPDIR symbol (see page 10) based on the current values in memory. *String_value* is the name of the setup. The setup name should not include the .SU extension. If the setup has been successfully created, @STATUS is set to 0; if there has been an error creating a new setup, @STATUS is set to 1.

IMPORTANT: If the specified setup already exists, the script will be aborted; thus, make sure that the setup does not exist or delete the setup using the REMOVE command before issuing the NEW command.

EXAMPLE:

```
remove "CIS"
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 terminal.

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 defined by the SETUPDIR symbol. *String_value* is the name of the setup. 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 OK 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 defined by the SETUPDIR symbol. *String_value* is the name of the setup. 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. A SET statement differs from the LET statement in that mathematical operations cannot be performed in a SET statement.

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 specifies the maximum number of characters to save into the variable. It must be greater than 0 and may be up to 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 specifies 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 TTRAPS. 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 = "ABCDABCDABCD"
strinx @string1, "A"           # look for pattern "A"
display "The letter A occurs first 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

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 = "ABCDABCDABCD"
strrinx @string1, "A"         # look for last occurrence of "A"
display "The letter A 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*.

EXAMPLE:

```
set @name = "Anemometer"  
strtrim @name, 4, 6  
display "Hi," , @name
```

SYMTYPE

reports the variable type

FORMAT: SYMTYPE *user-defined variable*

SYMTYPE determines the type (NONE, BINARY, STRING) of *user-defined 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 @data
```

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 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 display has been turned on in the script.

EXAMPLE:

```
display "Script paused..."
terminal
display "Script continuing..."
```

TRAPNULLS OFF

disable null traps

FORMAT: *TRAPNULLS_OFF*

TRAPNULLS_OFF disables the trapping of nulls; disabling null traps is the default mode.

EXAMPLE:

```
trapnulls_off
```


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 eq "0" write "Timeout on trap"
if @STATUS eq "1" write "Connected!"
if @STATUS eq "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 178.

EXAMPLE:

```
connect
tsend "lse SAL.TXT", CR # Send cmd to start lse on remote
wait 3
tsend "^z"                # Put lse in cmd mode
tsend "goto bottom", CR # Move cursor to end of file
tsend "^z"                # Take lse out of cmd mode
tupload "SAL.TXT"
wait 3 idle
tsend "^z"                # Put lse in command mode
wait 1
tsend "exit", CR         # Exit lse
ttrap 30, "SAL.TXT"     # trap filename - exit status line
set @hold = @status
wait 3 idle
if @hold eq "0"
    display "Tupload not completed; error ", @hold
    return
end
else display "Tupload successful"
```

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 15

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.

@7BITCHNread/write
YES [NO]

For BLAST protocol transfers, defines the data-path width.

BLAST Protocol subwindow: 7-Bit Channel

@ACKFREQread/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 265).

BLAST Protocol subwindow: Ack Request Frequency

@APROTOread/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

@ARGnread only
user-defined

Stores an “argument” (string value) passed from the operating system command line (see *argument* on page 11). The *n* specifies the argument, from 0 to 9 (@ARG0 stores the first argument on the command line, @ARG1 stores the second, @ARG2 stores the third, etc.).

@ATTKEYread/write
any Control Key [^K]

Defines the attention key (*ATTN*). Setting this variable to null (set @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

@AUTOLFINread/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

@AUTOLFOUTread/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

@BAUDRATEread/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 by the BLASTDIR symbol (see “Assigning Symbol Values” on page 7).

@CHARDLYread/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 name
hunt filename NONE

Stores the specification for the communications port or hunt file that BLAST will use for the current session. Valid options are:

Device name – Any valid asynchronous port (e.g., `ttal1:`).

Hunt filename – The name (including path) of a hunt file that lists available devices preceded by the “<” character. Refer to “Automatic Port Searching” on page 17 for details about hunt files.

NONE – Setting `@COMMPORT` to `NONE` allows the user to run scripts without opening a communications port. Commands requiring an open communications port, such as `FILETRANSFER` and `TERMINAL`, will not be allowed with `@COMMPORT` set to `NONE`.

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

Included for compatibility with other versions of BLAST. Function has not been implemented.

@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 the -dd or -y switch. 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 [%m/%d/%y]

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 294) 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 eq "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
[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

@ENBLERCMD

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

@FILBKSB read/write
[0] – 63

For indexed binary files, specifies the bucket size in 512-byte blocks.

Default VMS File Attributes subwindow: Bucket Size

@FILBKST read/write
[0] – 63

For indexed text files, specifies the bucket size in 512-byte blocks.

Default VMS File Attributes subwindow: Bucket Size

@FILECNT read-only

Returns the number of bytes either written or read during FREAD, FWRITE, FREADB, and FWRITEB.

@FILFSZB read/write
[0] – 255

For binary files, specifies in bytes the size of the control area for VFC files.

Default VMS File Attributes subwindow: Control Area Size

@FILFSZT read/write
[0] – 255

For text files, specifies in bytes the size of the control area for VFC files.

Default VMS File Attributes subwindow: Control Area Size

@FILLRLB read/write
0 – 32240 [512]

For binary files, specifies in bytes the size of the record.

Default VMS File Attributes subwindow: Record Size

@FILLRLT read/write
0 – 32240 **[4096]**

For text files, specifies in bytes the size of the record.

Default VMS File Attributes subwindow: Record Size

@FILMRSB read/write
[0] – 32767

For binary files, specifies in bytes the maximum record length.

Default VMS File Attributes subwindow: Max Record Length

@FILMRST read/write
[0] – 32767

For text files, specifies in bytes the maximum record length.

Default VMS File Attributes subwindow: Max Record Length

@FILORGB read/write
[SEQ] REL IDX

For binary files, specifies the RMS (Record Management System) file format. SEQ specifies Sequential, REL specifies Relative, and IDX specifies Indexed.

Default VMS File Attributes subwindow: File Organization

@FILORGT read/write
[SEQ] REL IDX

For text files, specifies the RMS (Record Management System) file format. SEQ specifies Sequential, REL specifies Relative, and IDX specifies Indexed.

Default VMS File Attributes subwindow: File Organization

@FILRATB read/write
[NONE] FTN PRN CR

For binary files, specifies the RMS (Record Management System) record attributes. Possible settings are:

NONE – None
FTN – Fortran
PRN – Print
CR – Carriage return/carriage control.

NOTE: If Record Format is set to STM, STMLF, or STMCR, and Record Attribute is set to NONE, the VMS system will change the Record Attribute to CR.

Default VMS File Attributes subwindow: Record Attribute

@FILRATT read/write
NONE FTN PRN [CR]

For text files, specifies the RMS (Record Management System) record attributes. Possible settings are:

NONE – None
FTN – Fortran
PRN – Print
CR – Carriage return/carriage control

NOTE: If Record Format is set to STM, STMLF, or STMCR, and Record Attribute is set to NONE, the VMS system will change the Record Attribute to CR.

Default VMS File Attributes subwindow: Record Attribute

@FILRFMB read/write
[UDF] FIX VAR VFC
STM STMLF STMCR

For binary files, specifies the RMS (Record Management System) record format. Possible settings are:

UDF – Undefined
FIX – Fixed
VAR – Variable
VFC – Variable length/fixed length control area
STM – Stream
STMLF – Stream/line feed
STMCR – Stream/carriage return

Default VMS File Attributes subwindow: Record Format

@FILRFMT

read/write
UDF FIX **[VAR]** VFC
STM STMLF STMCR

For text files, specifies the RMS (Record Management System) record format. Possible settings are:

UDF – Undefined
FIX – Fixed
VAR – Variable
VFC – Variable length/fixed length control area
STM – Stream
STMLF – Stream/line feed
STMCR – Stream/carriage return

Default VMS File Attributes subwindow: Record Format

@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

@FILXBKB

read/write
[YES] NO

For binary files, specifies whether a record can extend beyond a block boundary.

Default VMS File Attributes subwindow: Records Span Block

@FILXBKT

read/write
[YES] NO

For text files, specifies whether a record can extend beyond a block boundary.

Default VMS File Attributes subwindow: Records Span Block

@FULLSCRread/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

@INACTIMOread/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

@KBCHECKread/write
1 – 3 **[2]**

For Kermit transfers, specifies the level of error-detection.

Kermit Protocol subwindow: Block-Check-Type

@KDELAYOSread/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

@KEYBOARDread/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.

@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 – ^_ [^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

@KRETRY read/write
1 – 99 [10]

For Kermit transfers, specifies the number of times Kermit will attempt to send a single packet before aborting.

Kermit Protocol subwindow: Retry Limit

@KRFILETYP read/write
[BINARY] TEXT

For Kermit transfers, specifies the type of file being received.

NOTE: The reserved variable @KFILETYP is still supported and can be used to set both @KSFILETYP and @KRFILETYP to TEXT or BINARY. If @KSFILETYP and @KRFILETYP are set to the

same values, a `DISPLAY` of `@KFILETYP` will show that value. If, however, `@KSFILETYP` and `@KRFILETYP` are set to the different values, a `DISPLAY` of `@KFILETYP` will return an error.

Kermit Protocol subwindow: Receive Transfer Type

@KRNAMCONV read/write
[YES] NO

For Kermit transfers, specifies whether to convert filenames from local format to common Kermit format. Lower case is changed to all uppercase; and “~”, “#”, and all periods after the initial one are converted to “x”s.

NOTE: The reserved variable `@KFNAMCONV` is still supported and can be used to set both `@KSNAMCONV` and `@KRNAMCONV` to ON or OFF. If `@KSNAMCONV` and `@KRNAMCONV` are set to the same values, a `DISPLAY` of `@KFNAMCONV` will show that value. If, however, `@KSNAMCONV` and `@KRNAMCONV` are set to the different values, a `DISPLAY` of `@KFNAMCONV` will return an error.

Kermit Protocol subwindow: Receive Filename Conversion

@KRPADCH read/write
[^@] – ^ _

For Kermit transfers, specifies an alternate character to pad each packet received.

Kermit Protocol subwindow: Pad Character

@KRADDNG read/write
[0] – 99

For Kermit transfers, specifies the number of padding characters to request per packet.

Kermit Protocol subwindow: Padding

@KRPKTAVG read-only

Reports the average packet size of the last Kermit file received.

@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] – ^_

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 – ^_ [^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

@KSFILETYP

read/write
[BINARY] TEXT

For Kermit transfers, specifies the type of file being sent.

NOTE: The reserved variable @KFILETYP is still supported and can be used to set both @KSFILETYP and @KRFILETYP to TEXT or BINARY. If @KSFILETYP and @KRFILETYP are set to the same values, a DISPLAY of @KFILETYP will show that value. If, however, @KSFILETYP and @KRFILETYP are set to the different values, a DISPLAY of @KFILETYP will return an error.

Kermit Protocol subwindow: Send Transfer Type

@KSNAMCONV

read/write
[YES] NO

For Kermit transfers, specifies whether to convert filenames from local format to common Kermit format. Lower case is changed to all uppercase; and “~”, “#”, and all periods after the initial one are converted to “x”s.

NOTE: The reserved variable @KFNAMCONV is still supported and can be used to set both @KSNAMCONV and @KRNAMCONV to ON or OFF. If @KSNAMCONV and @KRNAMCONV are set to the same values, a DISPLAY of @KFNAMCONV will show that value. If, however, @KSNAMCONV and @KRNAMCONV are set to the different values, a DISPLAY of @KFNAMCONV will return an error.

Kermit Protocol subwindow: Send Filename Conversion

@KSPADCH

read/write
[^@]–^_

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

@KSPKTAGV read-only

Reports the average packet size of the last Kermit file sent.

@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] – ^_

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

@KSTIMEOUT read/write
0 – 99 **[10]**

For Kermit transfers, specifies the number of seconds that the computer will wait after transmitting a packet before attempting to re-send it.

Kermit Protocol subwindow: Timeout

@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 is 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 190). 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 reserved variable **@DATEFORMAT**.

@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 ex-

iting 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 library program SYSTEMS.SCR 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 10.

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

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 only after entering Filetransfer mode.

@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 at-

tempt 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.

@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 current 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. This is valid only after entering File-transfer mode.

@RSITE read-only

For BLAST protocol, stores the BLAST site number of the remote system. This is valid only after entering Filetransfer mode.

@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 defined by the SETUPDIR symbol (see “Assigning Symbol Values” 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.

@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 current 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.

LErrorR – Send not completed, due to local error.

RerrorR – 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.

@SSUCCESS 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 193.

@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.

@SYMBOLTYPE

read-only

Returns the results of the last SYMTYPE command—NONE, BINARY, or STRING.

@SYDESC

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 [%H:%M:%S]

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. The area is active if @TRANSTAT is set to ON. 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.

@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 something 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 @TRPASSWD to something other than 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.

@VMSFILESWread/write
[NO] YES

For BLAST protocol, appends all relevant VMS-specific file transfer switches (see “VMS-Specific Switches with BLAST Protocol” on page 91) to names of files sent to another VMS system.

IMPORTANT: If you set this field to YES for transfers to a system other than VMS, you may get an error message.

BLASTscript Protocol subwindow: Append VMS File Switches

@WDWSIZread/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 acknowledgment from the remote. As packets are acknowledged, the start point of the window is adjusted, or “slides.” See “BLAST Protocol Design” on page 79 for a fuller discussion of window size.

BLAST Protocol subwindow: Window Size

@WT4ECHOread/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

@XCRCread/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 on the following page. When Extended Logging is enabled, all the values from the variables listed are in the log file except for those 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
@SLINEQ	@RLINEQ	@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 274).

Setup field: Translate File

@XONXOFF

read/write
YES [NO]

Specifies whether software flow control is enabled. Not all computers support XON/XOFF flow control.

@XPADC

read/write
any character in decimal [00]

For Xmodem transfers, specifies the pad character. This parameter may also be set with the `-p` command line switch (“Command Line Switches” on page 10).

XYmodem Protocol subwindow: Pad character

@XYCONVRread/write
ASCII **[BINARY]**

For Xmodem and Ymodem transfers, specifies whether received files will be treated as ASCII or BINARY.

XYmodem Protocol subwindow: File Conversion

@XYCONVSread/write
ASCII **[BINARY]**

For Xmodem and Ymodem transfers, specifies whether files sent will be treated as ASCII or BINARY.

XYmodem Protocol subwindow: File Conversion

@XEOTread/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

@XYRLTRread/write
[CR/LF] CR LF

For Xmodem and Ymodem transfers, specifies how line termination is treated if @XYCONVR is set to ASCII.

CR/LF – lines of text are terminated by a carriage return followed by a line feed (CR/LF); for example, when ASCII files are received from a DOS or Windows platform.

CR – lines of text are terminated by a carriage return (CR); for example, when ASCII files are received from a Macintosh platform.

LF – lines of text are terminated by a line feed (LF); for example, when ASCII files are received from a UNIX platform.

XYmodem Protocol subwindow: Remote Line Termination

@XYRLTSread/write
[CR/LF] CR LF

For Xmodem and Ymodem transfers, specifies how line termination is treated if @XYCONVS is set to ASCII.

CR/LF – lines of text are terminated by a carriage return followed by a line feed (CR/LF); for example, when ASCII files are sent to a DOS or Windows platform.

CR – lines of text are terminated by a carriage return (CR); for example, when ASCII files are sent to a Macintosh platform.

LF – lines of text are terminated by a line feed (LF); for example, when ASCII files are sent to a UNIX platform.

XYmodem Protocol subwindow: Remote Line Termination

@YSTRIPread/write
YES [NO]

For sending files with Ymodem, specifies that the path and version number be stripped from the filename. This feature can prevent file transfer failures or incorporation of directory names into the remote filename due to system incompatibility.

XYmodem Protocol subwindow: Send Stripped Filename

@ZMALTread/write
[CR/LF] LF

For sending ASCII files to nonstandard implementations of Zmodem, specifies line-feed conversion for ASCII files. When @ZMCONVS = "ASCII", CR/LF terminates lines with CR/LF; LF terminates lines with LF.

Zmodem Protocol subwindow: ASCII Line Termination

@ZMAUTODOWNread/write
YES [NO]

For Zmodem transfers, specifies Auto Receive mode, which begins downloading immediately after entering Filetransfer mode.

Zmodem Protocol subwindow: Auto Receive

@ZMBLKLNread/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

@ZMCONVRread/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

@ZMCONVSread/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

@ZMCRCread/write
16 BITS **[32 BITS]**

For Zmodem transfers, specifies which CRC error-detection is to be used.

Zmodem Protocol subwindow: CRC

@ZMCTLESCRread/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

@ZMCTLESCSread/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

@ZMEXISTread/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

@ZMFRMLENread/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

@ZMMANAGRread/write
NONE PROTECT
[CLOBBER] APPEND

For Zmodem transfers, specifies a file management option for files received. See the File Management setup field on page 72 for a description of each option.

Zmodem Protocol subwindow: File Management

@ZMMANAGSread/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 70 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

@ZMSTRIP

read/write
YES **[NO]**

For sending files with Zmodem, specifies that the path and version number be stripped from the filename. This feature can prevent file transfer failures or incorporation of directory names into the remote filename due to system incompatibility.

Zmodem Protocol subwindow: Send Stripped Filename

@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 16

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.

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 55) 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,	CE,	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:

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

BLAST Professional VMS provides two terminal types, TTY emulator and PASSTHRU:

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 “Hot Keys”

on page 300 and “Attention Key Sequences” on page 300, 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.

Chapter 17

Limited Control of a Remote PC

If the remote computer is a PC running BLAST BHOST, you may connect to the remote, transfer files via BLAST protocol, and act as a text-based display terminal to the remote PC.

Connecting to the Host PC

Connecting to the remote PC, called 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. Nearly all of the configuration settings 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. 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.
- ◇ Set Protocol to BLAST.
- ◇ Set Packet Size to at least 200, BHOST's minimum 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 **File Transfer Only** and **Terminal**.

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 **τ** 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” on page 284.

Transferring Files to and from the Host PC

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 two 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 Terminal mode — Press **ESC CTRL X** to start **Filetransfer** mode on the Host PC; then press **F** to start **Filetransfer** mode locally.

Transferring Files

You may transfer files interactively (see “Performing Filetransfer Commands” on page 85) or via **BLAST** scripts (see “File Transfers with **BLAST** Protocol” on page 165).

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 Terminal mode. Otherwise, you will be returned to the Online menu.

Disconnecting from the Host PC

From File Transfer Only mode – Press `ESC` to return to the Online menu. After the inactivity timeout period, you will be automatically disconnected from the remote computer.

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 Terminal Mode

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

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:

<u>PC Key</u>	<u>Escape Sequence</u>
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
Numeric Keypad +	ESC +
Numeric Keypad -	ESC -
F1–F10	ESC 1 – ESC 0
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:

<u>PC Key</u>	<u>Escape Sequence</u>
Filetransfer mode	ESC CTRL X
Repaint Screen	ESC CTRL R
Open Session Command window	ESC CTRL M
Log off	ESC CTRL L

Modifying BHOST Settings

If you have a Terminal BHOST account, you may alter the BHOST session parameters via the Session Command window (Figure 17-1, 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 param-

eter. 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:

<u>Parameter Command</u>	<u>Parameter</u>
SCALE	Scaling Ratio
SYNC	Sync Mode
SCAN	Scan Interval
PMOUSE	Precision Mouse (unsupported; set to OFF)
SKEYBOARD	Special KBD Handling
INACTIMO	Inactivity T/O
TIMORESP	Timeout Response
DCDRESP	DCD Loss Response
MOUSE	Host Mouse
SCREEN	Host Screen
KEYBOARD	Host Keyboard
PRINTER	Host Printer
PRINT	Printer(s) Enabled

To close the Session Command window, press ESC.

```
BLAST Host Session Command Mode
>settings

Current BLAST Host Settings

SCALE=1:1
SYNC=ON
SCAN=MEDIUM
PMOUSE=OFF
SKEYBOARD=ON
INACTIMO=120
TIMORESP=RESTART
DCDRESP=RESTART
MOUSE=ON
SCREEN=ON
KEYBOARD=ON
PRINTER=NONE
PRINT=NONE
```

FIGURE 17-1

Following is a description of each Session Parameter. The default setting is in bold and brackets.

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 quickly and image quality is not important.

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.

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.

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 it 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 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 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 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.

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 58 for details.) |
| 22 | console interrupt
the ATTN key was typed |

- 23 inactivity timeout
(A BLAST protocol session was terminated because of inactivity. See the Inactivity Timeout setup field description on page 59 for details.)
- 24 error in processing command file
- 25 can't connect to the 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.)
----	---

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	cannot extract TERM
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	usage 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 comput-

er 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	invalid value for reserved variable
301-399	syntax error in command
400	too many strings

Network

502	fatal network error; BHOST terminated
-----	---------------------------------------

Appendix B

Key Definition Charts

Setup Keys

The following keys are for navigating through and editing a setup:

<u>Function</u>	<u>Key</u>
Cursor Up	↑ or CTRL E
Cursor Down	↓ or CTRL X
Cursor Left	←
Cursor Right	→
Move to First Field	CTRL R
Move to Last Field	CTRL C
Clear text in Field	CTRL T
Toggle Select/Edit Mode	CTRL P
Toggle Insert/Overwrite Mode	CTRL V [†]
Move to start of line	CTRL F [†]
Move to end of line	CTRL G [†]
Delete or Preceding Multiple-Choice Option	BACKSPACE
<i>CANCEL</i>	ESC or CTRL A ^{††}
<i>HELP</i>	? or F1 ^{††}

[†] For use in setup field Edit mode only.

^{††} Also for use outside of setups.

Attention Key Sequences

Attention Key sequences (shown below) are only active from Terminal mode. Online Help is only available when Terminal emulation is set to TTY. The Attention key can be redefined by entering a new setting in the Attention Key setup field (page 55).

<u>Function</u>	<u>Key Sequence</u>
Return to the Online menu	<i>ATTN ATTN</i>
Display Online Help	<i>ATTN H</i>

Hot Keys

BLAST features Hot Keys for accessing certain functions from Terminal when Terminal emulation is set to TTY (Hot Keys will not function in PASSTHRU mode). Hot Keys are essentially macros that activate BLAST menu commands and return you to your starting point with just a few keystrokes.

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.

<u>Function</u>	<u>Key</u>
Filetransfer	CTRL F
Local System	CTRL N
Learn	CTRL R

Appendix C

Troubleshooting

1. **“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.

2. **“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 BLAST symbols must be properly defined to allow the user to execute BLAST. For more information on BLAST symbols, see “Assigning Symbol Values” on page 7. 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” on page 176).

3. “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 “Port Parameters for BLAST in Host Mode” on page 18).

4. “What’s a quick way to get started with scripting?”

Use BLAST’s Learn mode (page 148) to build a script as you go through the steps of a process interactively.

5. “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.

6. “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 184).

7. “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	lf	<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 nine 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 SENDs.
AUTOPRCV.SCR – checks log for status of GETs.
AUTOPARX.SCR – updates status files.
AUTOSW.SCR – strips file transfer switches off @filename.

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:

```
set default [user.blast]
create /dir [user.blast.poll]
copy auto*scr [user.blast.poll]
delete auto*scr;*
```

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 **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]`

The command line parameters have the following meaning:

<code>autopoll</code>	the autopoll setup.
<code>-sautopoll.scr</code>	the autopoll script.
<code>max_cycles</code>	the maximum number of attempts to complete all specified transfers.
<code>site_file</code>	the filename “stub” (the part of the filename before the extension) of the site description file.
<code>[start_time]</code>	[optional] the time, in 24-hour format, that Autopoll will begin polling. If this parameter is omitted, Autopoll begins polling immediately.
<code>[TRACE]</code>	[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`, and `BLAST` will suppress its terminal output (`-n`). 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

<i>setup_name</i>	specifies a setup to be used for polling. If omitted, the field defaults to autopoll.
<i>site_name</i>	contains a descriptive label for the site. If omitted, the field defaults to the Description field of <i>setup_name</i> .
<i>phone_number</i>	specifies the phone number to be used for the site. If omitted, Autopoll uses the Phone Number field of <i>setup_name</i> .
<i>baud_rate</i>	specifies the baud rate to be used for this site. If omitted, Autopoll uses the Baud Rate field of <i>setup_name</i> .
<i>TCF_name</i>	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 and is not included in the site file):

```
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 94.

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 the script 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 313) 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 @SYSDDESC, @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 317 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 VMS system and two remote VMS 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:

<u>Site name</u>	<u>Phone</u>	<u>Login, password</u>
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 46).

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
|...|...|...|...|...|...|...|...|...|...|
+[USER.BUZ]ACQ12.TXT [USER.CLIENT] ]SAM1
+[USER.BUZ]WEEKLY_82 [USER.CLIENT] ]SAM2
[USER.TMP]MESSAGE [USER.TMP]READ_ME/OVW
```

As explained in "Transfer Command File" on page 94, the "+" sign in column 1 of a line of a transfer command file signifies that BLAST will perform a GET. Thus, in the file SAMS.TCF above, BLAST will get the file [USER.BUZ]ACQ12.TXT and give it the

local filename [USER.CLIENT]SAM1. BLAST will also get the file [USER.BUZ]WEEKLY_82 and give it the local filename [USER.CLIENT]SAM2. The absence of a “+” in the last line of the TCF signifies that BLAST will perform a SEND. Thus, BLAST will send the file [USER.TMP]MESSAGE and give it the filename [USER.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 with BLAST Protocol” on page 89 for more information about filetransfer switches).

METRO.TCF is similar to SAMS.TCF:

```

1           10           20           30           40           50
|...|...|...|...|...|...|...|...|...|...|
+[USER.NEIL]ACQ12.TXT [USER.CLIENT]METRO1
+[USER.NEIL]WEEKLY_82 [USER.CLIENT]METRO2
[USER.TMP]MESSAGE [USER.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 as follows:

<u>Extension</u>	<u>Created by</u>	<u>Meaning</u>	<u>Example</u>
.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. Since the contents of the file are irrelevant, the **create** command is a convenient way to create a stop file. In the Autopoll directory at the command line, type:

```
create retail.stp ENTER
CTRL Z
```

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 this 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 confirms a successful polling session. The files are saved in the Autopoll directory.

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 10.8.1 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.
*****
```

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.
```

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 symbol (see BPRINTER on 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 avoids the need for wildcard sends; it extracts the names of text files from a directory and from those filenames builds a TCF to send the files to a remote site.

```
# AUTOUSR0.SCR
#
# This script extracts the names of text files from a
# directory and uses the names to create a TCF. Without the
# user having to know specific filenames, the script
# creates a TCF that will transfer to the remote system all
# the text files from a specific directory, thus avoiding
# the use of wildcards, which prohibits file transfer
# retries.
#
# The script returns the following errors:
#
# 0 - no error
# 1 - cannot delete file
# 2 - cannot create/open file
# 3 - directory or files not found
#
if EXIST "FILES.LOG"
    ldelete "FILES.LOG"
    if NOT OK return 1
end
set @LOGFILE = "FILES.LOG"
lolist long "[STORES.INV]*.TXT" # capture dir listing in log
if @STATUS LT "1"
    set @LOGFILE = ""
    return 3 # dir or files not found - abort
end
set @LOGFILE = ""
if EXIST "AUTO.LOG"
    ldelete "AUTO.LOG"
    if NOT OK return 1
end
set @LOGFILE = "AUTO.LOG"
set @ustring1 = ".TXT" # set strings to find filenames
set @ustring2 = "[STORES"
fopenr 2, "FILES.LOG"
if NOT OK
```

```

    set @LOGFILE = ""
    return 2
end
if EXIST "INV.TCF"
    ldelete "INV.TCF"
    if NOT OK
        set @LOGFILE = ""
        fclose 2
        return 1
    end
end
fopena 3, "INV.TCF" # create TCF to write to
if not OK
    fclose 2
    set @LOGFILE = ""
    return 2
end
.SEARCHLOOP
fread 2, @uline # read log file & extract filenames
if OK
    strinx @uline, @ustring1
    if @STATUS not EQ "0"
        let @uposend = @STATUS + "3"
        strinx @uline, @ustring2
        let @upostart = @STATUS
        set @ufname = @uline
        strtrim @ufname, @upostart, @uposend
        # write filenames to TCF:
        fwrite 3, @ufname, " [STORES.NEWINV]", %
    end
    goto .SEARCHLOOP
end
fclose 2
fclose 3
ltype "INV.TCF" # Type TCF file for testing
set @LOGFILE = ""
return

```

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.

<u>Site</u>	<u>Name</u>	<u>Phone</u>	<u>Modem Type</u>	<u>Port</u>	<u>BLAST Version</u>	<u>System Type</u>
-------------	-------------	--------------	-------------------	-------------	----------------------	--------------------

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, Auto-poll loads the setup AUTOPOLL.SU by default.

<u>Site</u>	<u>Name</u>	<u>Setup Name</u>
-------------	-------------	-------------------

- 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

<u>Setup</u>	<u>Name</u>	<u>Phone</u>	<u>Baud</u>	<u>TCF</u>
--------------	-------------	--------------	-------------	------------

- 1.
- 2.
- 3.

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.

<u>Site</u>	<u>S/G</u>	<u>Remote Name</u>	<u>Local Name</u>	<u>Options</u>
-------------	------------	--------------------	-------------------	----------------

- 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 BLAST Symbols

Check the values of BLASTDIR, SETUPDIR, 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 20). 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

/APP

- BLAST Protocol 90
- Kermit Protocol 106
- Xmodem Protocol 114
- Zmodem Protocol 119

/BKSIZE

- BLAST Protocol 92, 116
- Kermit Protocol 108

/COMP=*n*

/FWD

- BLAST Protocol 90
- Enabling/Disabling 62, 100, 240
- Xmodem Protocol 114
- Ymodem Protocol 117
- Zmodem Protocol 119

/GROUP=*nn*

- BLAST protocol 90
- Kermit Protocol 106
- Xmodem Protocol 114

/LRECL

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 116

/MAXREC

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 116

/ORG

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 115

/OVW

- BLAST Protocol 90
- Enabling/Disabling 62, 240
- Xmodem Protocol 114
- Zmodem Protocol 119

/OWNER=*nn*

- BLAST Protocol 90
- Xmodem Protocol 114

/PERMS=*nnnn*

- BLAST Protocol 90–91
- Kermit Protocol 107

- Permissions Rules 127
- Xmodem Protocol 114–115

/RATTR

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 116

/RTYPE

- BLAST Protocol 92, 115
- Kermit Protocol 108

/STR

- BLAST Protocol 91
- Enabling/Disabling 62, 100, 240
- Xmodem Protocol 115

/TXT

- BLAST Protocol 91
- Xmodem Protocol 115
- Ymodem Protocol 117
- Zmodem Protocol 119

/VFCSIZE

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 116

/XBLK

- BLAST Protocol 92
- Kermit Protocol 108
- Xmodem Protocol 116

@STATUS 262

- Commands Set by 193
- Saving Value of 155–156

A

ASCII

- Character Set 303
- Control Codes 304
- Script Command 194

Attention Key. *See* *ATTN* Key

ATTN Key 26, 36

- Aborting Scripts 144
- Sequences 300
- Setup Field 55

Automation

- BLAST Protocol 98
- Scripting 44
- See also* Autopoll

Autopoll 305–322

- Banner File 314
- Command Line 306–308

- Configuration 311–313, 321–322
- Installing 306
- Modifying 317–320
- Remote Commands 309
- Setup 306–307, 308, 311
- Site File 308–309, 312
- Starting 306–308, 313
- Stop File 314
- Summary Files 315–316
- Tips 316–317
- Transfer Command File 309, 312–313
- User-Supplied Scripts 317–320
- Wildcards 309

B

BANNERTIME 8

Batch Mode 12

Baud Rate

- Compression Level and 99

- Hunt File Setting 18

- Reserved Variable 235

- Setup Field 53

- Site File and 308–309

BHOST 281–291

- BLAST Setup 282

- Compression Level 282

- Login 282

- Modifying Settings 285–291

- Packet Size 73, 282

- Starting BLAST 82–83

- Transferring Files 283–284

- See also* Control of Remote PC *and*

- BHOST Settings

BHOST Settings 285–291

- DCD Loss Response 289–290

- Host Keyboard 290

- Host Mouse 290

- Host Printer 291

- Host Screen 290

- Inactivity T/O 288–289

- Printer(s) Enabled 291

- Scaling Ratio 287

- Scan Interval 288

- Special KBD Handling 288

- Sync Mode 287

- Timeout Response 289

Binary Data Manipulation 193

Binary Variables, Defined 192

BLAST

- Batch Mode 12

- Host Mode 12–13, 281–291

- Quiet Mode 13

- Screen 24–26

- Starting 23–24

- Symbol for Executable 8–9

- Symbols 7–10

- Unattended 305–322

BLAST Protocol 77–101

- Advantages 78–79

- Automating 98

- Circuit Requirements 80

- Compression Level 98–99

- CRC Error Detection 79

- Design 79–80

- Ending File Transfer 84–85

- Extended Logging 13, 266

- File Transfer 85–101, 165–168

- File Transfer Switches 89–93

- File Transfer Templates 88–89

- Filename Restrictions 93

- Filetransfer Menu 40–41

- Fine-Tuning 98–99

- Getting Files 88, 165–166

- Message 41, 86, 167, 203

- Packet Acknowledgement 61, 80, 234

- Packet Size 15–16, 80, 98, 253

- Remote Commands 166–167, 204

- Remote Menu 86, 97–98

- Restarting Interrupted Transfer 94

- Scripting Considerations 167–168

- Security 100–101, 125–141

- Sending Files 87–88, 165–166

- Setup Subwindow 58–63

- Starting File Transfer 81–84

- Timeouts 83–84

- See also main entry* Timeout

- Transfer Command File 94–97, 167

- Transfer Options 86–87

- Transfer Password 61–62, 100–101, 264

- Wildcards 88

- Window Size 59–60, 265

- XON/XOFF Pacing 20

BLAST Session Protocol. *See* BLAST Protocol

BLASTDIR 7–9

Blaster (Online Demonstration and Testing Service) 28–34

- Connecting to 30–31
- File Transfer 31–33
- Logging Off 34
- Setup 28
- Terminal Compatibility 31

BLASTscript. *See* Script Commands and Scripting

BLPASSWD 128–135

- Header Information 130–132
- Record Information 132–135
- Symbol for 9

BLSECURE 135–139

- Symbol for 9

BPRINTER 9

C

CALL Statement 156–157, 195–196

CANCEL Key 26, 36, 37

Capture 39–40, 123, 225–226

Command Area 24–25

Command Line Arguments 11, 164–165

- Autopoll 306–308
- BLSECURE 136–139

Command Line Switches 10–14

- ? 13
- argument* 11, 164–165, 306–308
- Autopoll 306–308
- b 12
- dd 12
- dt 12
- e 12
- h 12–13, 176–177, 301–302
- n 13, 162, 245–246, 259
- p 13
- q 13
- s 11, 144, 164
- setupname* 11
- v 13
- x 13, 266
- Xmodem Protocol 112, 176–177
- y 14
- Ymodem Protocol 112, 176–177
- Zmodem Protocol 112, 176–177

Communication with Other Programs 164–165

Compression Level

- BHOST 282
- BLAST Protocol 98–99
- Reserved Variables 236, 255, 259
- Setup Fields 62–63

CONNECT Statement 147, 197

Connecting 147, 181–184

Connection Timeout 51

Control of Remote PC 281–291

- Connecting to Host PC 281–283
- Disconnecting from Host PC 284
- File Transfer Only Mode 283
- Terminal Mode 283, 284–285

CRC Error Detection 79

D

Data Stream

- Alteration 273–277
- Control 273–279
- Filtering 273–277
- Substitution 276
- Translate File 274–277
- Translation 276

Date Format

- @DATE 237
- @DATEFORMAT 237–238
- @LOGDATEFORMAT 251
- dd 12

DCD Loss Response 60, 168, 289–290

Default Setup 46

Default Values for Reserved Variables 233

Demo Line. *See* Blaster

Demonstration Service. *See* Blaster

Disconnecting 181–182, 184–185

DISPLAY Statement 163–164, 198

Documentation 3–4

Downloading Text 123, 180

E

Echo

- Local 56, 251, 302
- PAD Parameter 324
- Password Security and 254
- Script Command 199
- Wait for 56, 265

Edit Command 42

- EDITOR 10
 - Emulation. *See* Terminal Emulation
 - Error Checking 177
 - Error Detection
 - CRC 72, 79, 269
 - Modem 21
 - Setup Field 68
 - XON/XOFF Pacing 21
 - Error Messages 293–298
 - BLAST Protocol Functions 293–294
 - BLSECURE 139
 - Command File Processing 296
 - Initialization 296–297
 - Network 297–298
 - Script Processor 297
 - Scripting 295–296
 - Transfer File Management 294–295
 - Utility File Management 295
 - Extended Logging 266
 - Command Line Switch 13
 - Reserved Variable 266
- F**
- File Attributes
 - File Transfer Switches 91–93, 107–108, 115–116
 - Reserved Variables 241–244, 265
 - Setup Fields 63, 73–75
 - File Transfer
 - BHOST 283–284
 - BLAST Protocol 85–101, 165–168
 - Blaster 31–33
 - Error Checking 177
 - Kermit Protocol 103–108, 168–171
 - Xmodem Protocol 113–116, 171–172
 - Ymodem Protocol 116–117, 172–174
 - Zmodem Protocol 118–119, 174–175
 - File Transfer Status Area 25–26
 - File Transfer Switches
 - BLAST Protocol 89–93
 - Enabling/Disabling 62, 240
 - Kermit Protocol 106–108
 - Security with 100
 - VMS-Specific 91–93, 107–108, 115–116
 - Xmodem Protocol 113–116
 - Ymodem Protocol 117
 - Zmodem Protocol 119
 - See also specific file transfer switches*
 - File Transfer Templates 88–89
 - Filename Restrictions 93
 - Filetransfer Menu 40–41
 - BLAST Protocol 85–86
 - Kermit Protocol 103–104
 - Xmodem Protocol 40
 - Ymodem Protocol 40
 - Zmodem Protocol 40
 - FILETRANSFER Statement 147–148, 200–206
 - See also* File Transfer
 - Filtering
 - Data Stream 273–277
 - VT Sequences 60, 244
 - Flow Control 19–21
 - Downloading Text 123
 - RTS/CTS Pacing 19
 - Uploading Text 121
 - XON/XOFF Pacing 20–21
- G**
- Global Variables, Defined 157
- H**
- HELP 26, 36, 37
 - Automatic Display 24
 - Context-Sensitive 25, 27, 37
 - Host Mode 12–13
 - See also* BHOST and Pseudohost
 - Hot Keys
 - Definition Chart 300
 - PASSTHRU and 278–279
 - Hunt File 17–18, 51, 306
- I**
- IF Statement 147, 148, 212–215
 - Inactivity Timeout
 - Reserved Variable 245
 - Setup Field 59
 - Index Utility 186–187
 - Symbol for 10

K

- Kermit Protocol 103–110
 - File Transfer 103–108, 168–171
 - File Transfer Switches 106–108
 - Filetransfer Menu 103–104
 - Packet Size 64, 248, 250
 - Receiving Files 105–106, 169–170
 - Remote Commands 109–110, 205–206
 - Remote Menu 104, 109–110
 - Sending Files 105, 169
 - Setup Subwindow 63–67
 - Timeout 65, 248
 - Versions 103
 - Wildcards 105, 106
 - XON/XOFF Pacing 20

Keyboard

- BHOST Settings 288, 290

Keys

- ATTN Key 36
- ATTN Key Sequences 300
- CANCEL Key 36, 37
- Definition Charts 299–300
- Frequently Used Keys 26, 36–37
- HELP Key 26
- Hot Keys 300
- Setup Keys 299

L

- Learn Mode 38, 148–150
- Local Commands 41–43, 202–203
- Local Menu 41–43
- Log File

- Error Checking 177
- Reserved Variables 239, 252
- Setup Field 54

Login

- BHOST 282
- Password 51, 254
- System Scripts 181, 183
- Userid 50, 264, 282

M

- Menus 35–44
 - Filetransfer 40–41, 85–86, 103–104
 - Local 41–43

- Navigation through 27, 35
 - Offline 37–39
 - Online 39–40
 - Remote 43–44, 97–98, 109–110
 - Summary of 27

Message 41

- Menu Command 86
- Script Command 167, 203, 206
- TCF Command 95–96

Modem

- DCD Signal 238, 289
- Error Detection 21
- Hunt File Setting 18
- Reserved Variable 252
- RTS/CTS Pacing 19
- Scripts 181, 184–185
- Setup Field 52
- Typical Settings 302
- XON/XOFF Pacing 20–21
- See also* Baud Rate

N

- Numeric Constant, Defined 190
- Numeric String, Defined 190
- Numeric Value, Defined 190

O

- Offline Menu 37–39
- Online Demonstration and Testing Service.
- See* Blaster
- Online Menu 39–40

P

- Packet Acknowledgement 80
 - Request Frequency 61, 234
 - Window Size 59–60, 265
- Packet Size
 - BHOST 73, 282
 - BLAST Protocol 15–16, 80, 98, 253
 - Kermit Protocol 64, 248, 250
 - Line Quality and 260
 - Setup Field 64, 73
- PADs 15–16
 - X.3 Standard Parameters 323–328

Parity
 7-Bit Operation and 80
 Blaster Setup Field 29
 PAD Parameter 328
 Reserved Variable 253
 Setup Field 53
 Troubleshooting and 302

PASSTHRU 278–279

Password
 File, Secure BLAST 127–141
 Reserved Variable 254
 Security 127–141, 254
 Setup Field 51
 See also Transfer Password

Permissions 126–127
 BLPASSWD 133–134
 BLSECURE 137–138
 See also /PERMS=*mmm*

Ports 14–19
 Automatic Searching 17–18
 Checking Permissions 16–17
 Connection Setup Field 51
 Flow Control 19–21
 PADs and 15–16
 Parameters for Host Mode 18–19
 Reserved Variable 236
 X.25 Communications 15–16

Printing
 Autopoll Banner Files 314
 Autopoll Summary Files 315, 317
 BHOST Settings 291
 BPRINTER 9
 Enabling Remote Printing 97
 Error Message 296
 Menu Command 42, 43
 Remote 43, 96, 97, 98, 204
 Script Command 96, 202, 204, 217–218
 TCF Command 96

Protocols
 Definition 77
 Reserved Variable 255
 Setup Field 58
 Setup Subwindows 58–72
 See also individual protocols: BLAST Protocol, Kermit Protocol, Xmodem Protocol, Ymodem Protocol, *and* Zmodem Protocol

Pseudohost 176–177

Q

Quiet Mode 13

R

Registration 1

Remote Commands
 Autopoll 309
 BLAST Protocol 97–98, 166–167, 204
 Enabling/Disabling 62, 240
 Kermit Protocol 109–110, 205–206
 TCF 96

Remote Line Termination 68

Remote Menu 43–44
 BLAST Protocol 86, 97–98
 Kermit Protocol 104, 109–110

Reserved Variables 233–271
 @7BITCHN 234
 @ACKFREQ 234
 @APROTO 234
 @ARG*n* 11, 234
 @ATTKEY 234
 @AUTOLFIN 235
 @AUTOLFOUT 235
 @BAUDRATE 235
 @BLASTDIR 235
 @CHARDLY 235
 @CLASS 235
 @COMMPORT 236
 @COMP_LVL 236
 @CONNTIMO 236
 @CONTIMO 236–237
 @CTS 237
 @D/S_BITS 237
 @DATE 237
 @DATEFORMAT 237–238
 @DCD 238
 @DCDLOSS 238
 @EFERROR 148, 238–239
 @EFLOG 239
 @EFLOGGING 240
 @ELAPTIME 240
 @EMULATE 240
 @ENABLEFS 240
 @ENABLERCMD 240
 @FILBKSB 241
 @FILBKST 241

@FILECNT	241	@LOGDATEFORMAT	251
@FILFSZB	241	@LOGFILE	177, 252
@FILFSZT	241	@LOGTIMEFORMAT	252
@FILLRLB	241	@LOGTIMO	252
@FILLRLT	242	@MODEM	252
@FILMRSB	242	@NUMDISC	252–253
@FILMRST	242	@ONERROR	146–147, 253
@FILORGB	242	@ORGANS	253
@FILOGRT	242	@PAKTSZ	15–16, 253
@FILRATB	242–243	@PARITY	253
@FILRATT	243	@PASSWORD	254
@FILRFMB	243	@PHONENO	254
@FILRFMT	244	@PROMPTCH	254
@FILTER	244	@PROTOCOL	255
@FILXBKB	244	@RBTOT	255
@FILXBKT	244	@RBYTES	255
@FULLSCR	245	@RCLASS	255
@INACTIMO	245	@RCOMP_LEV	255
@KBCHECK	245	@RETRAN	255–256
@KDELAYOS	245	@RFAILURE	256
@KEYBOARD	245–246	@RLINEQ	256
@KFILETYP	246	@RNAME	256
@KFNAMCONV	246	@ROPTIONS	256
@KREOPKT	246	@RPACK	256
@KRETRY	246	@RPTOT	256
@KRFILETYP	246–247	@RRET	257
@KRNAMCONV	247	@RRTOT	257
@KRPADCH	247	@RSERIAL	257
@KRPADNG	247	@RSITE	257
@KRPKTAVG	247	@RSIZE	257
@KRPKTLEN	248	@RSTART	257
@KRSOPKT	248	@RSTATUS	257–258
@KRTIMEOUT	248	@RSUCCESS	258
@KSAVEINC	248	@RTIME	258
@KSEOPKT	248	@RTSCTS	19, 258
@KSFILETYP	249	@SBTOT	258
@KSNAMCONV	249	@SBYTES	258
@KSPADCH	249	@SCOMP_LEV	259
@KSPADNG	249	@SCRFILE	259
@KSPKTAVG	250	@SCRIPTERR	259
@KSPKTLEN	250	@SCRLREG	163, 259
@KSSOPKT	250	@SERIAL	259
@KSTIMEOUT	250	@SETUPDIR	259
@KWARNING	250	@SFAILURE	260
@LAUNCHST	251	@SITE	260
@LINEDLY	251	@SLINEQ	260
@LOCECHO	251	@SNAME	260

- @SOPTIONS 260
- @SPACK 260
- @SPTOT 260
- @SRET 261
- @SRTOT 261
- @SSIZE 261
- @SSTART 261
- @SSTATUS 261
- @SSUCCESS 261
- @STATUS 155–156, 193, 262
- @STIME 262
- @SYMBOLTYPE 262
- @SYSDISC 262
- @SYSTYPE 263
- @TIME 263
- @TIMEFORMAT 263
- @TRANSTAT 163, 263
- @TRAPCNT 263
- @TRPASSWD 264
- @TTIME 264
- @USERID 264
- @USERIF 163, 264
- @VERSION 265
- @VMSFILESW 265
- @WDWSIZ 265
- @WT4ECHO 265
- @XCRC 265
- @XLOG 266
- @XLTFILE 266
- @XONXOFF 20, 266
- @XPADC 266
- @XYCONVR 267
- @XYCONVS 267
- @XEOT 267
- @XYRLTR 267
- @XYRLTS 268
- @YSTRIP 268
- @ZMALT 268
- @ZMAUTODOWN 118, 174–175, 268
- @ZMBLKLN 269
- @ZMCONVR 269
- @ZMCONVS 269
- @ZMCRC 269
- @ZMCTLESCR 269
- @ZMCTLESCS 270
- @ZMEXIST 270
- @ZMFRMLN 270

- @ZMMANAGR 270
- @ZMMANAGS 270
- @ZMRESUME 271
- @ZMSTRIP 271
- @ZMWINDO 271
- RTS/CTS Pacing 19
 - Reserved Variable 258
 - Setup Field 54

S

Screen

- Command Area 24–25
- Description of 24–26
- File Transfer Status Area 25–26
- Host PC 290
- Scrolling Region 25, 163–164
- Script Commands 189–231
 - ASCII 194
 - ASK 194–195
 - BIN2HEX 195
 - CALL 156–157, 195–196
 - CHECKSUM 196
 - CLEAR 196
 - CLEOL 197
 - CONNECT 147, 197
 - CURSOR 197
 - DISCONNECT 148, 198
 - DISPLAY 145, 163–164, 198
 - DROP 198
 - ECHO 199
 - ERRSTR 199–200
 - FCLOSE 161–162, 200
 - FILETRANSFER FILE 200
 - FILETRANSFER GET/SEND 201
 - FILETRANSFER LOCAL 202–203
 - FILETRANSFER MESSAGE 203
 - FILETRANSFER REMOTE 204–206
 - FLUSH 206
 - FOPENA 161, 207
 - FOPENR 161–162, 207
 - FOPENW 161–162, 208
 - FREAD 161–162, 208
 - FREADB 209
 - FREE 209
 - FREWIND 209
 - FWRITE 161–162, 210
 - FWRITEB 210

GETENV 211
 GOTO 157, 211
 HEX2BIN 211
 IF 147, 212–213
 IF-ELSE 213–214
 IF-END 148, 214
 IF-END/ELSE-END 214–215
 LCHDIR 215
 LDELETE 215
 LET 215–216
 LLIST 216
 LOAD 216
 LOCAL SYSTEM 165, 217
 LOWER 217
 LPRINT 217–218
 LRENAME 218
 LTYPE 218
 MENU 218
 NEW 219
 PUT 219
 PWD 219
 QUIT 220
 RAISE 220
 REMOVE 220
 REPS 157–158, 221
 RETURN 221
 SAVE 221
 SELECT 222
 SET 222, 233
 SETTRAP 160–161, 222–223
 STRCAT 158–160, 223
 STRINX 158–159, 223–224
 STRLEN 159, 224
 STRRINX 159, 224
 STRTRIM 159, 224–225
 SYMTYPE 225
 TCAPTURE 160–161, 225–226
 TERMINAL 226
 That Set @STATUS 193
 TRAPNULLS_OFF 226
 TRAPNULLS_ON 227
 TSEND 149, 227
 TSENDBIN 227
 TTRAP 149, 228
 TUPLOAD 178–179, 228–229
 UPPER 229
 WAIT 229
 WAIT CARRIER 230
 WAIT IDLE 230
 WAIT UNTIL 230–231
 WERROR 163, 231
 WRITE 163, 231
 Script File
 Reserved Variable 259
 Setup Field 54
 Scripting 143–180
 Automation with 44, 305–322
 Blank Lines in 154–155, 166, 192
 CALL Statement 156–157, 195–196
 Capturing Text 160–161
 Comments in 145, 192
 Communication with Other Programs 164–165
 CONNECT Statement 147, 197
 Data Types 189–192
 Downloading Text 180
 Error Checking 150, 177
 FILETRANSFER Statement 147–148, 200–206
 IF Statement 147, 148, 212–215
 Labels 145, 154
 Learn Mode 148–150
 Legal and Illegal Expressions 154–155
 Loop in 157–158
 Messages 95–96, 167, 203
 Programming Style 153–154
 Reading Files 161–162
 Remote Commands 96, 166–167, 204–206
 Sample 145–150
 Screen Display 162–164
 Syntax Rules 192
 Text Manipulation 158–162
 Text Transfers 178–180
 Transfer Command File 94–97, 167
 Uploading Text 178–179
 Writing Files 161–162
 See also Script Commands, Scripting File Transfers, *and* Scripts
 Scripting File Transfers 165–177
 BLAST Protocol 165–168
 Error Checking 177
 Kermit Protocol 168–171
 Pseudohost 176–177

- Xmodem Protocol 171–172
- Ymodem Protocol 172–174
- Zmodem Protocol 174–175
- See also* Script Commands, Scripting, and Scripts
- Scripts
 - Aborting 144
 - Index Utility 186–187
 - Invoking 143–144
 - Modem 181, 184–185
 - Slave 82, 100
 - System 181, 182–183
 - Writing 144–151
 - See also* Script Commands, Scripting, and Scripting File Transfers
- Scrolling Region 25
 - Display Control 163, 259
 - Displaying Text 163–164
- SECURE 139–140
 - Symbol for 10
- Secure BLAST 127–141
 - BLPASSWD 128–135
 - BLSECURE 135–139
 - Password File 127–141
 - SECURE 139–140
 - See also* Security
- Security 125–141
 - @PASSWORD and 254
 - BLAST Protocol 100–101
 - Permissions 126–127
 - See also* Secure BLAST
- Serial Ports. *See* Ports
- Session Command Window 285–291
- Setup 45–75
 - Autopoll 306–307, 308, 311
 - BHOST 282
 - Blaster 28
 - Creating 46–47
 - Default 46
 - Directory 46
 - Keys 299
 - Loading 46
 - Modifying 47–48
 - Removing 48
 - Window, Described 47–48
 - See also* Setup Fields and Setup Subwindows
- Setup Fields 48–75
 - 7-Bit Channel 59
 - ACK Request Frequency 61
 - Append VMS File Switches 63
 - ASCII Line Termination 69
 - Attention Key 55
 - Auto Receive 72, 118
 - AutoLF In 56
 - AutoLF Out 56
 - Baud Rate 53
 - Block-Check-Type 66
 - Bucket Size 75
 - Character Delay 57
 - Connection 51
 - Connection T/O 51
 - Control Area Size 75
 - Conversion Override 69
 - CRC 72
 - Data/Stop Bits 53
 - DCD Loss Response 60
 - Delay 66
 - Description 49
 - Emulation 55
 - Enable /FWD and /STR 62
 - Enable /OVW and Remote Cmds 62
 - End-of-Packet Char 64
 - EOT Timeout 67
 - Error Detection 68
 - Esc All Control Chars 70–71
 - File Attributes 73–75
 - File Conversion 68, 72
 - File Management 72
 - File Must Already Exist 69
 - File Organization 74
 - Filename Conversion 65–66
 - Filtering 60
 - Full Screen 55
 - Inactivity Timeout 59
 - Incomplete File 67
 - Launch String 61
 - Limit Block Length 71
 - Limit Frame Length 71
 - Line Delay 57
 - Local Echo 56
 - Log File 54
 - Logon T/O 58
 - Management Option 70

- Maximum Record Length 75
 - Modem Type 52
 - Number of Disconnect Blocks 61
 - Originate/Answer 51
 - Packet Size 15–16, 64, 73
 - Pad Character 64–65, 67
 - Padding 65
 - Parity 53
 - Password 51
 - Phone Number 49
 - Prompt Char 57
 - Protocol 58
 - Receive Compression Level 63
 - Record Attribute 74
 - Record Format 74
 - Record Size 75
 - Records Span Block 75
 - Remote Line Termination 68
 - Resume Interrupted File 69
 - Retransmit Timer 60
 - Retry Limit 66
 - RTS/CTS Pacing 19, 54
 - Script File 54
 - Send Compression Level 62
 - Send Stripped Filename 68, 70
 - Size of Tx Window 71
 - Start-of-Packet Char 63–64
 - System Type 49–50
 - Timeout 65
 - Transfer Password 61–62
 - Transfer Type 66
 - Translate File 55, 274, 277
 - Use “A” Protocol 60
 - Userid 50
 - Wait for Echo 56
 - Warning 67
 - Window Size 59–60
 - XON/XOFF Pacing 20, 53–54
 - See also* Setup and Setup Subwindows
 - Setup Subwindows 48
 - BLAST Protocol 58–63
 - Default VMS File Attributes 73–75
 - Kermit Protocol 63–67
 - Xmodem and Ymodem Protocol 67–68
 - Zmodem Protocol 69–72
 - Setup Window. *See* Setup
 - SETUPDIR 8, 10
 - Slave Script 82, 100
 - Sliding-Window Design 79
 - Starting BLAST 23–24
 - String Constant, Defined 190–191
 - String Values, Defined 191
 - Symbols
 - Assigning Values 7–10
 - BANNERTIME 8
 - BLAST 8–9
 - BLASTDIR 7–9
 - BLPASSWD 9
 - BLSECURE 9
 - BPRINTER 9
 - EDITOR 10
 - INDEX 10
 - SECURE 10
 - SETUPDIR 8, 10
 - System Scripts 181, 182–183
- ## T
- Technical Support 5
 - Terminal Emulation 277–279
 - PASSTHRU 278–279
 - Reserved Variable 240
 - Setup Field 55
 - TTY 277–278
 - Terminal Mode 39
 - BHOST 283, 284–285
 - Hot Keys 278–279
 - Local Echo 56, 251
 - Script Command 226
 - Terminals
 - Standard BLAST 277–279
 - See also* Terminal Emulation and Terminal Mode
 - Testing Service. *See* Blaster
 - Text Transfers 121–123
 - Downloading Text 123, 180
 - Scripting 178–180
 - Uploading Text 121–122, 178–179
 - Text Translation 277
 - Time Format
 - @LOGTIMEFORMAT 252
 - @TIME 263
 - @TIMEFORMAT 263
 - dt 12

Timeout

- BHOST Settings 288–289
- BLAST Protocol 83–84
- Connection 51, 236
- Inactivity 59, 245
- Kermit Protocol 65, 248
- Logon 58, 252
- Xmodem Protocol 67, 267
- Ymodem Protocol 67, 267
- Transfer Command File (TCF) 94–97, 167
 - Autopoll 309, 312–313
 - Message Command 95–96
 - Remote Commands 96
- Transfer Password 100–101
 - Reserved Variable 264
 - Setup Field 61–62
- Translate File 274–277
 - Reserved Variable 266
 - Setup Field 55
- Troubleshooting 301–302

U

- Uploading Text 121–122
 - Error Detection 121
 - Flow Control 121
 - Scripting 178–179, 228–229
 - Upload Menu Command 40

V

- Variables
 - Defined 190
 - See also* Reserved Variables
- VMS File Attributes. *See* File Attributes

W

- Wildcards 88
 - Autopoll 309
 - BLAST Protocol 88
 - Kermit Protocol 105, 106
 - Pseudohost Mode 176
 - Ymodem Protocol 117
 - Zmodem Protocol 118

X

- Xmodem Protocol 112–116
 - Command Line Switches 112, 176–177
 - Connection Restriction 113
 - File Transfer 112–116, 171–172
 - File Transfer Switches 113–116
 - Filetransfer Menu 40
 - Limitations 111
 - Pseudohost 176–177
 - Receiving Files 113, 172
 - Sending Files 113, 171
 - Setup Subwindow 67–68
 - Timeout 67, 267
 - XON/XOFF Pacing 20
- XON/XOFF Pacing 20–21
 - End-to-End 21
 - Individual Protocols and 20
 - Local 21
 - PAD Parameter 325, 327
 - PASSTHRU and 278
 - Problems with 20
 - Reserved Variable 266
 - Setup Field 53–54

Y

- Ymodem Protocol 116–117
 - Command Line Switches 112, 176–177
 - File Transfer 116–117, 172–174
 - File Transfer Switches 117
 - Filetransfer Menu 40
 - Limitations 111
 - Pseudohost 176–177
 - Receiving Files 117, 173
 - Sending Files 117, 173
 - Setup Subwindow 67–68
 - Timeout 67, 267
 - Wildcards 117
 - XON/XOFF Pacing 20

Z

- Zmodem Protocol 118–119
 - Auto Receive 72, 118, 174–175, 268
 - Command Line Switches 112, 176–177
 - File Transfer 118–119, 174–175
 - File Transfer Switches 119

Filetransfer Menu 40
Limitations 111
Pseudohost 176–177
Receiving Files 118, 175
Sending Files 118, 174
Setup Subwindow 69–72
Wildcards 118
XON/XOFF Pacing 20

TO: BLAST Technical Support

FAX #: 919-542-0161

FROM:

Voice #:

COMPANY:

FAX #:

DATE:

IMPORTANT: 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 the Documentation?

We would like to hear your feedback on the usefulness of this documentation. Your opinions can help us improve it in the future.

BLAST Professional VMS User Manual

2MNPVMS

October 2001

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.