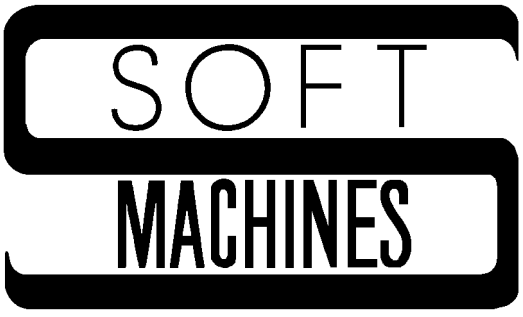
Autolog User's Guide



Printed in the United States of America

Third (electronic only) edition, September 2002

Soft Machines
PO Box 14
Sidney, IL 61877

(217) 688-3317
support@softmach.com

Contents

Introduction

Autolog Communications Package is a point-to-point data communications and file transfer software package. You can use Autolog to place calls using a modem, transfer files with other systems, or have your computer talk to other types of equipment or peripherals, such as bar code scanners, handheld computers, or printers.

You may be eager to get started right now, not read manuals. If Autolog is already installed on your computer, feel free to plunge in right away. Chapter 1 will get you started.

If you'd rather become more familiar with Autolog before getting started, read through this ***Autolog User's Guide***. This will tell you everything you need to know about how Autolog works.

If you're already familiar with Autolog on another type of computer system, you'll be pleased to know that everything should be very familiar. The commands will work very similarly or identically to those you've already learned. The ***Autolog Installation and Platform Guide*** will explain any operating system–specific variations in how commands work.

Look for notes marked with this symbol. They contain information about ways that Autolog works differently under different operating systems or using different types of equipment.

If Autolog hasn't been installed yet, refer to the ***Autolog Installation and Platform Guide*** for installation instructions.

Conventions Used in this Manual

So that you can easily tell what characters you need to enter to perform a task, and what is an example of the response you might see, we use these different styles:

`command` Characters you will type to enter an Autolog command or perform some other task. After typing an Autolog command, you must press the "Return" key (which may be called "Enter" or "Execute" on your keyboard) to execute the command. Autolog commands can be abbreviated to the fewest unique number of letters; for instance, the `receive` command may be entered as **`rec`** (enough letters to distinguish it from the `redial` command, which can be entered as **`red`**).

argument — The argument of a command gives specific information to complete the command. For example, to transfer a file, you will need to tell Autolog the name of the file(s) you want to transfer. We will show a sample file transfer command like this: **`ztransmit`** *file(s)*. You will type **`ztransmit`** and then the actual names of the files you want to send. When an argument is indicated as *true or false*, you may use **`true`**, **`t`**, **`on`**, or **`1`** to turn the option on, or **`false`**, **`f`**, **`off`**, or **`0`** (zero) to turn the option off.

`response` — Messages or characters displayed on your monitor by Autolog or by the computer.

key — A key you must press, such as the "Return" or "Enter" key (which will be indicated **Enter**), or the escape **ESC** key.

Autolog uses a number of special "hot keys" to do certain tasks. Because you can assign whatever keystroke you like to these keys, we will usually refer to them by name, such as the **copy key**, rather than by the actual keystroke you may use.

Control characters, which you enter by holding down the "control" or "ctrl" key while simultaneously pressing another key, will be indicated like this: **^J** for control+J.

cancel key — The key that is used by your computer to cancel or abort a command or program. This is usually control-C for AMOS and for DOS and Windows computers. For UNIX computers, the **cancel key** is defined by your shell: It may be control-C or Delete, or any other key defined by your shell. Refer to the ***Autolog Installation and Platform Guide*** for more information.

term — Words that appear in boldface are defined in the glossary in Appendix C.

Chapter 1
Getting Started

❖❖❖

1.1 Starting and Ending Autolog

To start the Autolog program, enter the command **`autolog`** (on DOS, UNIX, or AIX computers) or **`autlog`** (on AMOS, UNIX, or AIX computers) followed by **Enter** (a carriage return or new-line character, which may be labeled "Return," "Enter," or "New line," or "↵" on your keyboard) at the system or shell prompt. Autolog may also be a choice on your shell or menu. On Windows computers, you can select "Autolog" from the Programs menu of the Start task bar, or you can double-click the Autolog.exe icon in the Autolog folder (usually on the C: drive). If you want to use any special options when starting Autolog (such as using switches or a script file in the start-up command), you must use the `autolog` command from a DOS prompt window.

When Autolog starts up, the **cursor** will be resting next to Autolog's command prompt, which looks like this: >. You may also see Autolog's command screen, which will be similar to that illustrated in Figure 1.1. When Autolog's command prompt is displayed, Autolog is ready for you to enter commands. After typing a command, press **Enter**, and Autolog will execute the command. Whenever Autolog is ready for you to enter commands at the > command prompt, you are in Autolog's **command mode**.

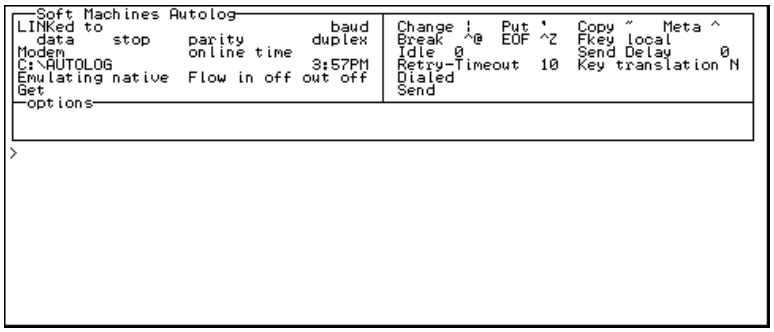


Figure 1.1 The Autolog command mode screen.

Optional Switches for the `autolog` Command

A number of optional switches can be used with the `autolog` command to control how Autolog behaves when you first start it up. The character you use to indicate a switch depends on your operating system: use a slash

(/) for AMOS and DOS or Windows, and a hyphen (-) for UNIX or AIX. See the ***Autolog Installation and Platform Guide*** for more information about using switches on your system.

The `mono` Switch

The `mono` switch will start Autolog in monochrome mode (not using color). Use this switch if you're using a monochrome display screen and the Autolog screen is hard to read.

The `t` Switch

The `t` switch will start Autolog in terse mode, which suppresses Autolog's formatted command mode display (see Figure 1.1). When you are in command mode while the `terse` option is on, you'll see only Autolog's command prompt, the > character. You can put Autolog into terse mode after it's running by using the `terse` command (discussed in Chapter 2).

The `n` Switch

The `n` switch will prevent Autolog from using its normal start-up script. The start-up script for your system may include tasks that Autolog routinely needs to perform before starting a communications session, such as selecting a modem or adjusting the baud rate. See Chapter 4 for more information about the start-up script, or `go`, file.

Advanced Options

When you first start Autolog, you can indicate the name of a script, or `go` file, to be used in place of Autolog's regular start-up script. Chapter 4 explains what `go` files are and how to create them. After you've created a `go` file, you can have Autolog use it when it first starts up by adding the `go` file name as an argument to your `autolog` command, for example:

```
autolog myscript
```

The preceding command would tell Autolog to perform the commands contained in the `go` file `myscript` *in place of* the normal start-up script.

You can also define macros when you first start up Autolog. Section 4.4 in Chapter 4 explains more about macros and how to specify them when you start up Autolog.

The `finish` Command

The first command you should know is `finish`. The `finish` command is used to exit Autolog, restore everything back to its normal state after using Autolog, and return you to your system prompt, shell, or menu. If your **modem** was connected, `finish` will hang up the modem. You will generally use `finish` to end every Autolog communications session.

The `quit` Command

Under special circumstances, you may not want everything restored to its pre-Autolog status. For example, if you adjusted the **baud rate** of your **communications port** and would like the port to remain at that baud rate until your system is rebooted or turned off, you may use the `quit` command to exit Autolog *without* restoring the port to its pre-Autolog state.

☞ `quit` will not hang up the modem.


On UNIX and AIX systems, some communication parameters may be returned to system defaults when you use the `quit` command.

1.2 General-Purpose Commands

Following are some commands that you may find useful throughout your communications session for navigation or for additional information while you're using Autolog.

The `system` Command

The `system` command returns you temporarily to system or shell level so that you may perform non-Autolog-related tasks such as erasing or editing files.

Enter the command **`system`** without an argument to temporarily access your system or shell, without disconnecting from your port or hanging up the modem.


The `system` command may not use the same shell that you normally use (the one you're running Autolog under). DOS and Windows systems will use COMMAND.COM, and UNIX and AIX systems will use the shell that the `system()` command uses.

When you're done with your system-level tasks and are ready to resume using Autolog, exit your shell.


AMOS users: Enter the command **`autlog`** to return to your Autolog communications session.

DOS/Windows users: Enter the command **`exit`** to return to your Autolog communications session.

UNIX and AIX users: Enter the command or keystroke used to exit your shell to return to your Autolog communications session.

☞ The `system` command does not hang up the modem or restore the communications port to its pre-Autolog state, so it is important that you remember to return to Autolog and use the `finish` command when you want to end your communications session.

You can specify a system-level command to perform a single command without exiting Autolog. Enter

`system` *command*

where *command* is the system-level command you want to use, in the same format as you would normally enter it at the system or shell prompt.

You can also use the shorthand command `!` for the `system` command when you include a system-level command as an argument:

`!` *command*

The `cd` Command

The `cd` command allows you to change your current working directory or the account you're currently logged to. Enter

`cd` *path*

where *path* is the directory you want to move to. The path should be specified in the way normal for your system. You may specify a relative path (based on your current working directory) or an absolute path (one that starts with a slash, a backslash, or a full disk specification, depending on your system).

The `direct` Command

The `direct` command displays a listing of the files in your current directory or account in a "wide" or multicolumn format. You may include switches that are normally honored by the "dir" or "ls" command on your system.

The `help` Command

The `help` command will display Autolog's online help. You can use Autolog's online help to find more information on any Autolog command.

If you get the error message ***`Help is not available`***, contact your system administrator. On some platforms, online help requires the use of an HTML browser, to which you may not have access on your system.

The `pwd` Command

The `pwd` command displays the current working directory or account.

The `type` Command

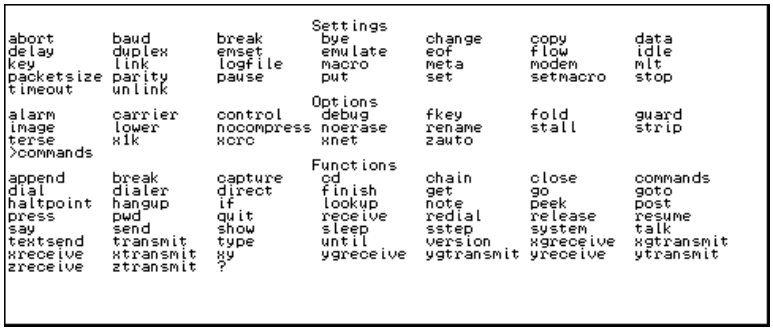
The `type` command displays the contents of the specified file on your screen. Enter

type *file*

where *file* is the name of the file whose contents you want to display. The display will pause after each screen of information; to see the next screenful of information, press any key (except the **cancel key**, **ESC**, or **Q**, which are used to cancel the display).

The commands Command

The commands command displays a list of all Autolog commands. The display will resemble that shown in Figure 1.2. You can also use the shorthand ? command to display a list of Autolog's commands.



```
                              Settings
abort       baud       break       bye        change     copy       data
delay       duplex     emset       emulate    eof        flow       idle
key         link       logfile     macro      meta       modem      mlt
packetsize  parity     pause       put        set        setmacro   stop
timeout     unlink
                              Options
alarm       carrier    control     debug      fkey       fold       guard
image       lower      nocompress  noerase    rename     stall      strip
terse       xlk        xcrc        xnet       zauto
>commands
                              Functions
append      break      capture     cd         chain      close      commands
dial        dialer     direct      finish     get        go         goto
haltpoint   hangup     if          lookup     note       peek       post
press       pwd        quit        receive    redial     release    resume
say         send       show        sleep      sstep      system     talk
textsend    transmit   type        until      version    xgreceive  xgtransmit
xreceive    xtransmit  xy          ygreceive  ygtransmit yreceive   ytransmit
zreceive    ztransmit  ?
```

Figure 1.2 commands display.

The version Command

The version command displays the version number and other useful information about your copy of Autolog. Use this command when calling for technical support or to find out whether you need to update Autolog.

1.3 Preparing Your Communications Port

To make a connection with a remote system, you will first need to tell Autolog where your modem is and to prepare your communications port and your modem.

The link Command

You will use the link command to tell Autolog which communications port your modem is connected to. Type the command

link *port*

where *port* is the specification of the communications port to which your modem is attached. If you don't know the name of your modem's port, refer to the ***Autolog Installation and Platform Guide*** for help or ask your

system administrator. Depending on the type of computer you use, the port may have a name like "tty2a," "COM2," or "MODEM1."

Enter the command **`link`** without an argument to display the available communications ports on your system.

The `unlink` Command

The `unlink` command unlinks Autolog from the current communications port and restores the port to its pre-Autolog status. Under normal circumstances, if you want to link to a new communications port, just issue a new `link` command. You don't need to `unlink` from one port before using `link` to connect to another.

The `baud` Command

If this is the first time you've used this modem or if you have an older modem that requires you to adjust the baud rate before you place a call, you'll need to use the `baud` command to adjust the baud rate of your communications port. Enter

`baud` *number*

where *number* indicates the baud rate at which you want to communicate with your modem. For newer modems, you'll generally need to set this baud rate only once, and you'll want to select the fastest baud rate your hardware can use successfully. For instance, **`baud 19200`** will set the communications port at 19200 baud, which should work well if you enable hardware flow control, discussed later.

For older modems, you may need to adjust the baud rate every time you want to place a call at a different baud rate. Some older modems will connect with the remote modem only at the baud rate at which the communications port is set. With this type of modem, you will set the baud rate with the `baud` command to the baud rate at which you expect the remote modem to answer. For instance, enter **`baud 2400`** to place a call at 2400 baud.

If you're unsure what type of modem you have, refer to your modem manual to find out if your modem supports independent "serial" and "modem" baud rates.

The `flow` Command

The `flow` command is used to activate flow control. **Hardware flow control** prevents loss of data during high-speed communications at high baud rates. You can specify a flow control signal for outgoing data (`flow out`) and for incoming data (`flow in`).

Use the command

`flow out` *signal*

where *signal* is **cts**, **dsr**, **xon**, or **off**, to select the hardware connection that the modem will use to signal your communications port when it is ready to receive outgoing data.

Not all of these choices are supported under all operating systems. If you try to select a `flow` setting that is not supported on your system, Autolog may display the error message *`interface driver does not support this feature`*.

For `cts` or `dsr`, the appropriate connection must be available in your modem cable. The `xon` option activates **software flow control** using the XON/XOFF characters. Software, or XON/XOFF, flow control is less efficient and less desirable than hardware flow control, and may prevent some file transfers (including XMODEM, YMODEM, and some ZMODEM) from working.

To select the flow control signal your computer will use to signal to your modem that it is ready to receive incoming data, use the command

`flow in` *signal*

where *signal* is one of these options: **rts**, **dtr**, **xon**, or **off**.

Again, not all of these choices are supported under all operating systems. If you try to select a `flow` setting that is not supported on your system, Autolog may display the error message *`interface driver does not support this feature`*.

For `rts` or `dtr`, the appropriate connection must be available in your modem cable. The `xon` option activates the less effective software flow control using XON/XOFF.

You should use this command only if your modem cable includes connections for hardware flow control. Also, some flow control options may not be available under different operating systems. Refer to the ***Autolog Installation and Platform Guide*** for information about your modem cable and the flow control options available on your computer.

The `data`, `parity`, and `stop` Commands

You may need to adjust the way serial data characters are constructed to suit the remote system. You generally need to do this before talking to your modem, so that your modem will recognize and use the correct settings. Autolog will use 8 data **bits**, no **parity**, and 1 stop bit unless you change these settings using the `data`, `parity`, and `stop` commands.

Autolog's default settings should work with most contemporary computer systems. However, some older computers and some mainframes or information services may require 7 data bits and even or odd parity, or some other configuration. The remote system operator or the information you receive for signing onto the remote system should tell you what to use

for these settings. This information may appear like this: “81N” (8 data bits, 1 stop bit, no parity) or “71E” (7 data bits, 1 stop bit, even parity).

Enter

`data` *number*
or
`stop` *number*

where *number* is the number of bits to use, to change the number of data bits or stop bits, respectively. Enter

`parity` *even, odd, or none*

to change the parity.

☞ 8 data bits are required by most error-correcting files transfers, including XMODEM, YMODEM, and ZMODEM.

The `modem` Command

Next, you’ll need to tell Autolog the type of modem you use. For modems that use the “AT” modem command set, Autolog can automatically dial and hang up your modem for you. Enter the command:

`modem at`

to tell Autolog that your modem understands the AT modem command set.

If your modem does not use AT commands, you will need to dial and hang up the modem yourself.

1.4 Placing a Call

If you have an AT-type modem, you can place a call using Autolog’s `dial` command. If you have a different type of modem, you must give the modem its dial command yourself. See the appropriate section below for your type of modem.

The `dial` command

The `dial` command tells Autolog to instruct your modem to dial a phone number. Enter the command:

`dial` *phone number*

where *phone number* is the number you want the modem to dial. Autolog will give the appropriate command to your modem to dial the number, then wait for then wait for the remote modem to answer.

You may include any special characters that your modem permits to perform special dialing functions. Here are some special dialing characters that Autolog can use if they are supported by your modem:

W waits until a second dial tone is detected before dialing the rest of the number.

, pauses for 2 seconds before dialing the rest of the number. You may insert as many **,**'s as necessary.

P pulse dials.

T tone dials.

? waits until you enter a character (anything except the **cancel key**) before continuing to dial. Autolog displays the following prompt on the screen:

```
Press any key to continue dialing:
```

This character is useful when dialing credit card authorization centers or certain voice mail systems that permit tone dialing for selecting extensions or for inputting information. The number to be dialed may contain as many ?'s as necessary.

Not all modems support the ? feature.

When the remote modem answers, you will see the message:

```
Entering talk mode...Press | to return to command mode
```

This indicates that you are ready to talk to the remote system. While you're in **talk mode**, characters you type are sent through your communications port and transmitted by your modem over the communications channel. Data received from the remote system are displayed on your screen in talk mode. You're ready to use the remote system! You may skip ahead to Section 1.5, "Communicating with the Remote System."

The `redial` Command

The `redial` command can be used to redial the phone number used in a previous `dial` command or can be used in place of the `dial` command. Enter

`redial` *times*

where *times* is the number of times you want to try dialing the number.

To use `redial` in place of the `dial` command, specify the phone number to dial after the number of times to try dialing:

`redial` *times phone_number*

For instance, **`redial 3 555-1212`**.

Placing a Call without the `dial` Command

If you don't have an AT-type modem, or if you didn't use the `modem at` command, you will have to dial your modem yourself. To enter talk mode and be able to "talk" to your modem, press the **change key**, which is

initially the vertical bar or "change bar" key |. After you press the **change key** you will see the message:

```
Entering talk mode...Press | to return to command mode
```

You are now ready to type the command to tell your modem to dial. Refer to your modem manual for help if you're unsure of the command you need to give the modem. If you have a very old modem, you may need to dial the telephone by hand. Now is the time to do this. After the remote modem answers, you can proceed to Section 1.5, "Communicating with the Remote System."

If you're using Autolog to talk to a piece of equipment other than a modem, you don't need to use the `dial` command either. Just press the **change key**, and when you see the message:

```
Entering talk mode...Press | to return to command mode
```

you're ready to talk to the remote device.

If the Call Is Not Completed

If the call is not answered by a remote modem, Autolog will display an appropriate error message. The error message you receive may depend on the types of errors your modem is capable of reporting. For instance, some modems report simply "no answer" if the line is busy or if there is no answer by a remote modem. Other modems can distinguish between these conditions. Autolog will report *`no answer`* or *`line is busy`*, depending on the error given by your modem.

If your modem is not working properly, Autolog will report *`modem does not respond`*. The problem may be that your modem is not turned on, is at the wrong baud rate for your communications port, hasn't been configured properly to work on your computer with Autolog, or has a problem that requires hardware repair.

The `hangup` Command

The `hangup` command hangs up your modem if you have an AT-type modem or if your modem hangs up when the DTR hardware signal is pulsed. Otherwise, you must manually hang up your modem or enter the appropriate "attention" signal and hangup command for your modem while in talk mode.

You only need to use the `hangup` command when you want to end the current phone call and place another without leaving Autolog. Autolog will normally hang up the modem automatically when you use the `finish` command to exit Autolog.

Refer to your modem manual and the ***Autolog Installation and Platform Guide*** to determine if `hangup` will hang up your modem. If you see an error message, *`modem does not respond`*, or if

the modem's off-hook (OH) light remains lit after entering the `hangup` command, Autolog could not hang up the modem. Enter your modem's hangup command while in talk mode, or disconnect the phone line from the modem to force the modem to hang up.

1.5 Communicating with the Remote System

When the remote modem answers, you're ready to access the remote system. After a successful `dial` command, Autolog will automatically enter talk mode, and you will see the message

```
Entering talk mode...Press | to return to command mode
```

If you had to manually dial the modem, press the **change key** (initially the vertical change bar, |) to enter talk mode.

After making a modem connection to the remote system, while you are in talk mode, characters sent to and from the remote system will be displayed on your terminal screen, just as if you were using the remote system directly. The keys you type at your keyboard will be sent to the remote system, and the remote system's responses will be displayed on your screen.

To give Autolog another command, you must reenter command mode by pressing the **change key**. You can toggle back and forth between command and talk modes by pressing the **change key** any time during your communications session.

You may find that, due to the differences between your computer and the remote system, some Autolog settings may have to be adjusted in order for you to use the remote system effectively. For instance, you may find that the remote computer doesn't echo what you type on your screen, so you need to use the `duplex` command. Making these types of adjustments to Autolog's settings is the subject of the next chapter.

1.6 Autolog's Reports

Autolog can keep a record of the phone calls placed using the `dial` command and can record a log file of your communications session.

Phone Log File

The Autolog phone file will contain the date and time of the call, the connect time, the name of the job or user who placed the call, and the phone number dialed. Figure 1.3 shows a sample phone log. See the ***Autolog Installation and Platform Guide*** for more information about phone log file options.

```
02-14-1997 13:25 Length 8:08:22 Selena 3517411
02-14-1997 14:55 Length 1:01:11 Karen 5551212
02-14-1997 15:00 Length 2:02:57 Karen 3517411
03-02-1997 14:38 Length 0:00:09 Selena 5551212
03-02-1997 14:53 Length 15:15:12 Selena 3517411
03-02-1997 14:55 Length 0:00:37 Selena 3517411
03-02-1997 14:55 Length 0:00:15 Karen 3517411
```

Figure 1.3 The Autolog phone log file.

You can add a note to the phone log file with the `note` command. Enter

`note phone` "*text*"

where *text* is the text you wish to add to the phone log file, enclosed in a pair of double quotes. The text will appear in the phone log file at the beginning of the line for that phone call entry.

Session Log File

You can also have Autolog keep a log file of your communications session with the `logfile` command. Enter

`logfile` *file*

where *file* is the name you want to give to the communications session log file. Autolog will record all talk mode activity and all commands entered in command mode into the log file. Figure 1.4 shows a sample log file.

☞ We recommend that you use terse mode (see Chapter 2) for log files that are succinct and easy to read.

You can add a note to the log file with the `note` command. Enter

`note log` "*text*"

where *text* is the text you want to add, enclosed in a pair of double quotes. The text will be added immediately to the log file.

```
>dial 3517411
Autolog talk mode, press \ to return to command
Entering talk mode, press \ to return to command mode

Enter name: selena
Enter password: *********
Welcome to Soft Machines MSP-based Update System

.dir
CH1    DOC 156   DSK2:[77,0]
X      X   8
Total of 2 files in 164 blocks
.
>receive y.y=x.x
Receiving X.X as Y.Y
      3947 bytes

      3947 bytes to go,     0 retries,     0 CRC errors
  0% complete  256 byte packets      3947 bytes to go,
0 retries,     0 CRC errors
 59% complete  571 byte packets         0 bytes to go,
0 retries,     0 CRC errors
100% complete  311 byte packets
Elapsed time was 0:00:01, effective transfer rate was
39470 bps.

Total transfer time was 0:00:01, effective transfer
rate was 39470 bps.
>
Releasing remote site from SMT mode.
Autolog talk mode, press \ to return to command mode
Entering talk mode, press \ to return to command mode
.logout
Logged out.

>f
```

Figure 1.4 Sample log file.

Chapter 2
Changing Settings

2.1 Local Settings and Settings to Suit the Remote System

You may need to adjust some settings after you've connected with the remote system. You may discover that things don't look the way you expect, function keys don't behave the way they should, or information is displayed incorrectly on the screen. These are all signs that you may need to use one of the commands discussed in this chapter. If you know in advance what settings will best suit the remote system, you can change them before you even connect. Otherwise, you may have to do a little experimentation or talk to the remote system operator to help you discover what settings will work best.

Not getting any response at all or getting only "garbage" when you connect to the remote system may indicate a more fundamental problem that must be corrected by adjusting the settings discussed in Chapter 1. A mismatch in baud rate, improper flow control, or incorrect serial character settings fall into this category. To correct these settings, you must hang up the modem, correct the settings, then reconnect. See Chapter 1 for more information.

You may want to change some local settings to make Autolog more convenient for you to use or to change how Autolog behaves to perform a special task. Examples of such local settings include redefining Autolog's "hot keys" to be ones that you prefer to use or settings that control how Autolog handles special characters after they're received from the remote system.

The "options" box on Autolog's command screen indicates which settings are currently active. For example, the "options" box in Figure 2.1 indicates that the guard option has been turned on.

In this chapter, we'll look at both remote and local settings. First we'll discuss settings you may need to change to suit the remote system, then we'll discuss commands to change how Autolog behaves locally.

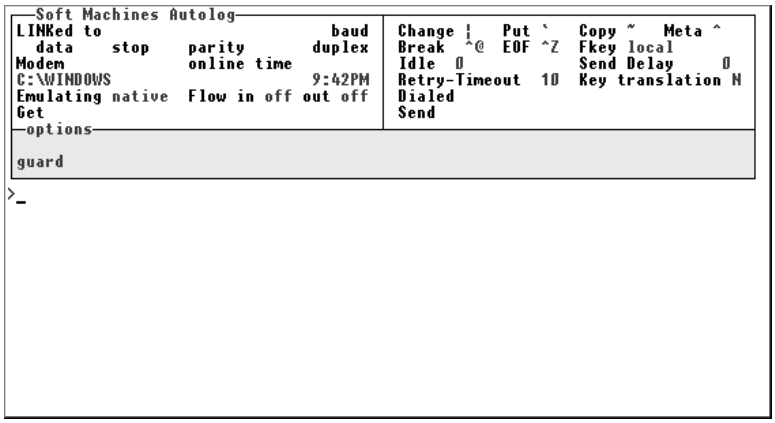


Figure 2.1 The "options" box of Autolog's command screen indicates the options that are turned on.

2.2 Adjusting Echoing with the `duplex` Command

The `duplex` command determines whether Autolog or the remote system will echo back the characters you type. Full duplex means the remote system will echo your characters. Half duplex means the remote system will not echo your characters, so you will probably want to make Autolog echo for you so you can see what you've typed. Enter the command

`duplex` *f, h, or s*

where *f* stands for full duplex, *h* for half duplex, or *s* for special half duplex. Special half duplex will echo the destructive backspace (erasing the previous character) when you type a rubout, backspace, or delete character. The information you received for signing on to the remote system may tell you the correct duplex to use. This information may appear like this: "81NF" (the final F tells you to use full duplex) or "71EH" (the final H indicates half duplex; the other characters refer to the serial character settings, discussed in Chapter 1).

When you receive responses from the remote system, but cannot see what you've typed, select **`duplex h`** or **`duplex s`**. If everything you type is echoed TTWWIICCEE, select **`duplex f`**.

2.3 Translating Keys with the `key` Command

The `key` command allows you to redefine any key on your keyboard for use in talk mode. You may find that one or two keys don't seem to work correctly on the remote system (the backspace or delete key is a notorious example), or you may need to send a special keypress for which you don't

have a key on your keyboard. The `key` command lets you specify what should be sent to the remote system when you press that key while in talk mode. You can use the `key` command to redefine function keys or to create your own custom "macro" keys.

To redefine a key, enter the command

`key set` *number* **`"`***string***`"`**

where *number* indicates which key on your keyboard to redefine (discussed below) and *string* (enclosed in quotes) is the character(s) you want that key to send to the remote system. The string can be one or more characters. For printable **ASCII** characters, just type the character(s) you want to assign to that key within a pair of quotes. For control characters, use the ^ symbol (or current **meta** character, discussed later in Section 2.6). For instance, control-C would be entered as **`"^C"`**. For non-ASCII characters, precede the decimal value of the character with the ^ symbol. For instance, the letter "u" (ASCII decimal value 117) with the high bit set would be **`"^245"`** (117 + 128). (See Appendix A for more information about ASCII characters and their decimal values.)

☞ A translated key sends its reassigned character(s) only while in talk mode. At any other time, it will send the character that is normal on your system.

Table 2.1 gives the decimal value of function keys and cursor-positioning keys for systems other than AMOS. To find out the appropriate number for other keys on your keyboard, use the command

`key test`

You will see the message

```
Press 3 successive space characters to exit key test
```

Now press the key(s) you want to redefine. As you press each key, the character it normally sends is displayed to the left and its numeric indicator is displayed to the right. If you press **F1** (the first function key), the display may look like this:

```
>key test
Press 3 successive space characters to exit key test
%5      160
Key     Number     "Translation"
```

Note the number to use in the `key set` command. The character(s) normally assigned to that key appear at the left above "Key." Control characters will be preceded by the symbol ^ (or the current **meta** character, discussed later in Section 2.6). Non-ASCII characters (including most function keys) will be preceded by the symbol %. If you have already redefined the key, the character(s) it currently sends while in talk mode will appear on the far right above "Translation." Press the spacebar three times to end the key test.

Table 2.1 Decimal Values of Function and Cursor Keys*

Key	Decimal value	Key	Decimal value	Key	Decimal value
F1	160	shift+F1	192	F13 (Alt+F3)†	172
F2	161	shift+F2	193	F14 (Alt+F4)†	173
F3	162	shift+F3	194	F15 (Alt+F5)†	174
F4	163	shift+F4	195	F16 (Alt+F6)†	175
F5	164	shift+F5	196	←	136
F6	165	shift+F6	197	↓	138
F7	166	shift+F7	198	↑	139
F8	167	shift+F8	199	→	140
F9	168	shift+F9	200	Insert	156
F10	169	shift+F10	201	Home	158
F11	170	shift+F11	202	Page Up	146
F12	171	shift+F12	203	Page Down	148
				End	133

* For systems other than AMOS.

† Alt+*function key* can be used on DOS and Windows computers whose keyboards lack function keys F13–F16.

Example. If you normally use the key labeled "backspace" or "←" to erase the previous character, but this key doesn't work on the remote system, you can redefine it to send the appropriate "erase" character. Many systems use the ASCII character DEL (decimal value 127) to perform the erase function, but others use the underline, _ (value 95). Still others use the ASCII BS character (control-H, value 8). Enter the command **`key test`**, then press your usual "erase" key to find out what ASCII character your system normally uses and to find out the number associated with that key. Press the spacebar three times to return to Autolog's command prompt. If your "erase" key normally sends the DEL character but the remote system requires an underline, for example, use the command **`key set`** *number* **`"_"`**. If your usual "erase" key sends the underline character, but the remote system uses the DEL character, use the command **`key set`** *number* **`"^127"`**. You may need to do a little experimenting on the remote system to discover what character it uses for "erase the last character."

You can save your key definitions in file that is loaded automatically each time you start up Autolog again using the same terminal or workstation. Enter the command

key save

Your key definitions will be stored in a file on disk that will be automatically reloaded when you start Autolog from the same terminal or workstation (using the same terminal emulation; see Section 2.5 below on the `emulate` command). The name of the key definition file will depend on the type of terminal or workstation you're using, your computer system, and whether or not terminal emulation is active.

You can give the key definition file a different name by using the command

key save *file*

where *file* is the name you want to give the stored key translation file. The next time you want to use those particular key translations, enter the command

go *file*

where *file* is the name of the stored key translation file. (See Chapter 4 for more information about the `go` command.)

After using the `key set` command, if you'd like to get rid of all the key translations and return to a "normal" keyboard, enter the command

key clear

This will clear all the key translations.

2.4 The `fkey` Command

The `fkey` command controls how function keypresses are transmitted and interpreted. Normally, function keys are translated by your local computer system, whether you're in command mode or talk mode. Your local function key processing may be inappropriate or meaningless on the remote system, in which case you may use the `fkey` command to have function keys processed by the remote system while in talk mode. Use the command

fkey remote

to have function keys processed by the remote system while in talk mode. (When in command mode, function keys will always be processed by the local computer.) Use the command

fkey local

to return to the default state in which function keys are processed by the local system at all times.

Some error-correcting modems and other communications equipment may not send multibyte function key sequences fast enough for them to be interpreted as function keypresses by the remote system. In this case, try turning off the modem's data compression or contact the modem manufacturer.

2.5 Terminal Emulation with `emulate` and `emset`

Different brands and models of terminals work differently. The remote system that you call may not be prepared to work with the type of terminal you're using. This will be apparent because screen displays won't look right: Text in columns may be run together or scrambled; certain characters may be missing or transformed into the wrong characters; video display styles such as reverse video, dim or bold, or underlining may not appear or may garble the text.

Autolog's `emulate` command lets you instruct your terminal to emulate, or act like, another type of terminal. You can emulate a VT-100™ terminal—a popular and widespread type of terminal from Digital Equipment Corporation supported by many remote systems—or any other ANSI terminal. The `emulate` command can also be used to emulate a teletype (TTY) terminal or the "generic" terminal AUTGEN.TDV available on many Alpha Micro™ computer systems.

Use the command

`emulate` *vt100, sco, vt52, bbs, tty, autgen, wyse60, or off*

to select the terminal emulation you want to use. Use **`emulate off`** to turn off a previously selected terminal emulator.

Many systems support VT-100 or VT-102 terminals, which will work with Autolog's `vt100` emulation. The `bbs` emulation will work well with systems that expect PC callers: It is an extension of the `vt100` emulation that also supports 8-bit PC graphics characters. The `bbs` emulation is recommended when calling bulletin board systems. The `sco` emulation is a special variant of the `vt100` emulation for use when calling SCO UNIX systems. The `vt52` emulation can be used with systems that expect callers to be using VT-52™ terminals, another Digital Equipment Corporation terminal less fully featured but similar in some respects to the VT-100. The `tty` emulation should be used when the remote system supports only a very primitive type of terminal known as a TTY or teletype. The `autgen` emulation should be used when communicating with an Alpha Micro computer that uses the AUTGEN "generic" terminal driver. The `wyse60` emulation can be used with systems that support the Wyse WY-60 terminal. The `wyse60` emulator works only with hidden (mode) attributes.

The remote system operator or support department for the remote system should be able to help you determine what types of terminals are supported and therefore the terminal emulation you should select with the `emulate` command.

The vt100 Emulations

Autolog's vt100 and bbs emulations are almost identical. The only difference between them is that the bbs emulation supports 8-bit PC graphics characters and is therefore useful when calling systems that expect you to be using a PC (when you're not). The following discussion, although it refers to the vt100 emulation, applies equally to Autolog's bbs emulation.

vt100 Emulation Video Displays

Because your terminal may lack some of the features of a VT-100, Autolog's vt100 emulation will display certain information slightly differently from a true VT-100 terminal.

Dim for bold. Autolog's vt100 emulation will use the dim video attribute in place of bold, because most terminals support the dim video display, but many do not support bold.

Printing functions. If you have a printer connected *directly* to your terminal (not a printer connected to your computer that you can access from your terminal), VT-100 printing functions will work as they do on an actual VT-100 terminal. Without such a printer, you can use the emset getmode command (discussed below) to redirect information that would normally be sent to the printer to a disk file instead.

Field terminals. Field terminals require an "invisible" character that takes up a space on the screen to change video display attributes such as dim or reverse video. The VT-100 is a mode terminal, which doesn't display this space. As an unfortunate but inevitable result, if the terminal you use is a field terminal, screen displays that use special video attributes may appear slightly out of alignment.

The emset Command

The VT-100 is a sophisticated terminal that can operate in many different modes. You can use the emset command to adjust the terminal emulation's settings to suit the remote system's VT-100 terminal support software. Enter the command

emset help

or

emset ?

to see a brief display of the settings that can be changed using the emset command, each of which is discussed next. Enter the command emset with no argument,

emset

to display the current emulation settings.

emset width. Use the `emset width` command to adjust the screen display to a width of 80 or 132 characters. Enter

emset width *132 or w*

to use a wide (132-column) display. Enter

emset width *80 or n*

to use a narrow (80-column) display.

emset answerback. The VT-100 can send an "answerback" in response to an ASCII ENQ character, for use with some electronic mail applications. To set your answerback, use the command:

emset answerback "*text***^M"**

where *text* is the character(s) of your answerback, enclosed in a pair of quotes. The characters `^M` at the end of the answerback text cause a carriage return or "enter" character (control-M) to be sent after the answerback text. You can include other control characters by preceding them with the **^** character (or current **meta** character, discussed later in Section 2.6).

emset getmode. The VT-100 terminal can have a printer connected directly to a port on the terminal. VT-100 control codes may send some information directly to the printer, instead of or in addition to being displayed on the screen. Use the `emset getmode` command to direct where such printer information is sent. If your terminal has a directly connected printer (not a printer connected to your computer that you can access from your terminal), you can have this information sent to your terminal's printer. Otherwise, you may direct this information to a disk file, using the `get` command (discussed in Chapter 3). Use the command

emset getmode *setting*

where *setting* is one of the following:

`printer` "Printer" data is sent to your terminal's directly connected printer.

`raw` This setting is the default `getmode`; it directs all printer data to a file on disk, which must be opened using the `get` command (see Chapter 3).

`text` Only text characters of the printer data are stored in the disk file (printable characters, carriage returns, and linefeeds). Other nonprintable characters are discarded.

`all` Nonprintable characters (such as screen formatting and cursor control commands) in the printer data are converted to a printable form, and written to the disk file along with the printable printer data. The terminal control codes Autolog uses appear in Appendix B.

☞ To use the `raw`, `text`, or `all getmode` settings, you must open a disk file using the `get` command as discussed in Chapter 3. The `get` function will then operate as usual (i.e., you may use the **copy key** to toggle on or off the capturing of data into the disk file).

`emset printkey.` The `emset printkey` command can be used to define a print key, which is used to "print" a current screen snapshot, analogous to the print key on the keyboard of an actual VT-100 terminal. With Autolog's `vt100` emulation, after you define a print key with the `emset printkey` command, each press of the print key will cause the current screen display to be recorded in the current `get` file, if one is open (see Section 3.6 of Chapter 3 for information on the `get` command). Enter the command

`emset printkey` "*key*"

where *key* is key you want to use as a print key, enclosed in a pair of double quotes. You can type a printable key directly, or use the `^` (or current `meta` character) to indicate control characters. For example, **`"^G"`** would make control-G the new print key. You can indicate a non-ASCII character (e.g., a function key) by entering **`"^`***number***`"`** where *number* is the decimal value of the key. For example, **`"^162"`** would make the F3 key (on systems other than AMOS) the current print key.

`emset local.` The `emset local` command can be used to change any `vt100` emulation setting supported by Autolog. Most of the `vt100` emulation settings that you will need to change can be changed by using one of the `emset` commands previously discussed. The `emset local` command lets you change settings that can't be changed by another `emset` command. Use the command

`emset local` "*escape sequence*"

where "*escape sequence*" is the VT-100 command sequence for altering the desired setting, enclosed in a pair of double quotes. The ANSI terminal standard, which is adhered to by the VT-100 terminal and other ANSI terminals, describes the **escape sequences** supported by ANSI terminals.

`emset reset.` The `emset reset` command resets the `vt100` emulation to its initial, default settings.

VT-100 Cursor and Keypad Keys

The VT-100 cursor, or arrow, keys and keypad keys can operate in two different modes. When the keypad keys are in *numeric mode,* they generate their normal numeric characters. But when they are in *alternate mode,* they act as function keys and generate multibyte function key sequences. Similarly, the cursor keys may be in *cursor mode,* in which

they generate multibyte cursor positioning commands, or *application mode,* in which they generate multibyte function key sequences. The VT-100 also has four function keys called PF1 (also called the *gold key*), PF2, PF3, and PF4.

If you are not using a VT-100 terminal locally, your terminal may lack some of these keys, or the characters they generate may not be the same as those the remote system expects to see. You may set up your own set of keys to use for these special VT-100 keys with the command

```
go vtkeys
```

You will then be prompted to press the key on your keyboard that you want to use for each of these VT-100 keys. Table 2.2 lists the special VT-100 keys for which you will be asked to select a key on your keyboard, along with characters that a VT-100 terminal generates when these keys are pressed. Notice that you will not be asked to assign a keypress to generate the characters in the gray areas of the table (the numeric keypad keys): For these keys you may use the regular number keys on your keyboard.

We recommend that you do *not* use your keypad keys for the VT-100 keypad keys unless your terminal generates keypresses for these keys that are distinct from your regular keyboard number keys (you can use `key test`, described above, to see if these keypresses are distinctive).

Some error-correcting modems or other communications equipment may not send multibyte sequences fast enough for them to be interpreted as function keypresses by the remote system. In this case, try turning off the modem's data compression or contact the equipment manufacturer.

The keys you assign VT-100 functions will generate the character sequences listed in Table 2.2 *only while in talk mode*. At any other time, they will generate the character(s) they normally do on your system.

The `sco` Terminal Emulation

The `sco` emulation causes your terminal to act like a SCO UNIX console using the `TERM=ansi` setting. The SCO UNIX `ansi` console is somewhat like a VT-100 terminal, but it supports 25 lines rather than 24 and uses private SCO control sequences for graphics characters and character colors. Use the `sco` emulation if you need to run `scoadmin`, `scosh`, or other programs that work best using the SCO UNIX console environment

Table 2.2 VT-100 Keypad and Cursor Keys

VT-100 keypad or cursor key	Corresponding key (selected with `go vtkeys`)	Numeric keypad mode/ cursor mode	ANSI/VT-100 alternate keypad/cursor application mode	VT-52 alternate keypad mode
0		0	ESC O p	ESC ? p
1		1	ESC O q	ESC ? q
2		2	ESC O r	ESC ? r
3		3	ESC O s	ESC ? s
4		4	ESC O t	ESC ? t
5		5	ESC O u	ESC ? u
6		6	ESC O v	ESC ? v
7		7	ESC O w	ESC ? w
8		8	ESC O x	ESC ? x
9		9	ESC O y	ESC ? y
–		–	ESC O m	ESC ? m
,		,	ESC O *l*	ESC ? *l*
.		.	ESC O n	ESC ? n
Enter		control-M	ESC O M	ESC ? M
↑		ESC [A	ESC O A	ESC A
↓		ESC [B	ESC O B	ESC B
→		ESC [C	ESC O C	ESC C
←		ESC [D	ESC O D	ESC D
PF1 (gold key)		ESC O P	ESC O P	ESC P
PF2		ESC O Q	ESC O Q	ESC Q
PF3		ESC O R	ESC O R	ESC R
PF4		ESC O S	ESC O S	ESC S

1 = numeral one, *l* = lowercase letter *l*.

The `sco` emulation supports a subset of the `emset` settings that are supported by the `vt100` emulation. These are the `emset` setttings that the `sco` emulation supports: `help` or `?`, `answerback`, `getmode`, `printkey`, `local`, and `reset`. See page 26 for information about the `emset` settings.

The `vt52` Terminal Emulation

The `vt52` emulation lets your terminal act like a VT-52 terminal. The VT-52 is similar to, but less sophisticated than, the VT-100 terminal. The VT-52 uses a subset of VT-100 commands. The VT-52 does not adhere to the ANSI terminal standard, and so the **escape sequences** that it requires

to control the screen and that it generates for special keys are different from those used by the VT-100 terminal.

The `vt52` terminal emulation uses the `emset` command to control certain settings. See the previous section on the `emset` command for details.

The `tty` Terminal Emulation

The `tty` terminal emulation can be used when calling systems that expect a TTY, or teletype, terminal. A teletype is a very primitive type of terminal that doesn't support any screen formatting. The only "screen control" or "escape sequences" it uses are the ASCII characters BEL (control-G, to sound a beep), BS (control-H, a nondestructive backspace), and DEL (erase the previous character).

The `tty` emulator does not use the `emset` command.

The `autgen` Terminal Emulation

The `autgen` terminal emulation should be used when calling an Alpha Micro computer that uses the AUTGEN "generic" terminal driver. The AUTGEN.TDV file was distributed with prior releases of Autolog for AMOS. If you would like more information about the generic terminal driver for Alpha Micro computers, please contact Soft Machines' technical support.

The `autgen` emulator does not require any additional settings, so the `emset` command is not used by this terminal emulator.

The `wyse60` Terminal Emulation

The `wyse60` emulation emulates the Wyse WY-60 terminal. The `wyse60` emulation works only with hidden (mode) attributes: attributes that don't take up a character space on the screen to display.

The `emset` Command

You can use the `emset` command to adjust the terminal emulation's setting to suit the remote system's WY-60 terminal support software or your own needs. Enter the command

`emset help`

or

`emset ?`

to see a brief display of the settings that can be changed using the `emset` command, each of which is discussed next. Enter the command `emset` with no argument,

`emset`

to display the current emulation settings.

emset width. Use the `emset width` command to adjust the screen display to a width of 80 or 132 characters. Enter

emset width *132 or w*

to use a wide (132-column) display. Enter

emset width *80 or n*

to use a narrow (80-column) display.

emset getmode. The WY-60 terminal can have a printer connected directly to an auxiliary port on the terminal. WY-60 control codes may send some information directly to the printer, instead of or in addition to being displayed on the screen. Use the `emset getmode` command to direct where such printer information is sent. If your terminal has a directly connected printer (not a printer connected to your computer that you can access from your terminal), you can have this information sent to your terminal's printer. Otherwise, you may direct this information to a disk file, using the `get` command (discussed in Chapter 3). Use the command

emset getmode *setting*

where *setting* is one of the following:

`printer` "Printer" data is sent to your terminal's directly connected printer.

`raw` This setting is the default `getmode`; it directs all printer data to a file on disk, which must be opened using the `get` command (see Chapter 3).

`text` Only text characters of the printer data are stored in the disk file (printable characters, carriage returns, and linefeeds). Other nonprintable characters are discarded.

`all` Nonprintable characters (such as screen formatting and cursor control commands) in the printer data are converted to a printable form, and written to the disk file along with the printable printer data. The terminal control codes Autolog uses appear in Appendix B.

☞ To use the `raw`, `text`, or `all getmode` settings, you must open a disk file using the `get` command as discussed in Chapter 3. The `get` function will then operate as usual (i.e., you may use the **copy key** to toggle on or off the capturing of data into the disk file).

emset printkey. The `emset printkey` command can be used to define a print key, which is used to "print" a current screen snapshot, analogous to the print key on the keyboard of an actual WY-60 terminal. With Autolog's `wyse60` emulation, after you define a print key with the `emset printkey` command, each press of the print key will cause the

current screen display to be recorded in the current `get` file, if one is open (see Section 3.6 of Chapter 3 for information on the `get` command). Enter the command

`emset printkey "`*key*`"`**

where *key* is key you want to use as a print key, enclosed in a pair of double quotes. You can type a printable key directly, or use the `^` (or current `meta` character) to indicate control characters. For example, **`"^G"`** would make control-G the new print key. You can indicate a non-ASCII character (e.g., a function key) by entering **`"^`***number*`"`** where *number* is the decimal value of the key. For example, **`"^162"`** would make the F3 key (on systems other than AMOS) the current print key.

`emset local.` The `emset local` command can be used to change any `wyse60` emulation setting supported by Autolog. Most of the `wyse60` emulation settings that you will need to change can be changed by using one of the `emset` commands previously discussed. The `emset local` command lets you change settings that can't be changed by another `emset` command. Use the command

`emset local "`*escape sequence*`"`**

where "*escape sequence*" is the WY-60 command sequence for altering the desired setting, enclosed in a pair of double quotes. Wyse documentation for the WY-60 terminal describes the control and **escape sequences** supported by WY-60 terminals.

`emset reset.` The `emset reset` command resets the `wyse60` emulation to its initial, default settings.

2.6 Redefining Autolog's Special Keys

We'll now look at local settings that can make Autolog more convenient for you to use or that can adjust how Autolog behaves in order to perform special tasks.

The first settings you may wish to change are Autolog's special keys. Autolog uses several "hot keys" to perform special functions (e.g., the **change key** used to toggle between command and talk modes). Table 2.3 lists these keys, along with their initial, or default, values and their functions. You can redefine any of these keys at any time while in command mode by using the procedures discussed in this section.

Table 2.3 Autolog's Special Keys

Key	Initially	Function
break key	none	Send **break** condition to the remote system.
change key	|	Toggle between talk and command modes.
copy key	~	Turn on or off recording to a disk file with `get` (see Chapter 3).
eof	control-Z	End-of-file character for `post` and `capture` (see Chapter 3).
meta	^	Indicates control characters.
pause key	none	Toggle on or off single-stepping of script files (see Chapter 4).
put key	‘	Turn on or off transmission of a file with `send` (see Chapter 3).

Type New Character as Argument

The first way to change one of Autolog's special keys is to enter the name of the key followed by the new character you want to use for that key. For example,

```
change +
```
or
```
put *
```

You may indicate control characters by using the ^ (or current **meta** character). For instance, to define the **break key** as control-B, enter

```
break ^B
```

Type Decimal Value as Argument

Nonprintable, non-ASCII characters may be entered using their decimal equivalent. For instance, to define the **copy key** as the letter "K" with the high bit set (K's ASCII decimal value is 75, so 75 + 128), enter

```
copy 203
```

Of course, to use a non-ASCII character, you must have some key on your keyboard, such as a function key, capable of producing that character. See Appendix A for the ASCII characters and their decimal equivalents.

Enter Key Directly at Prompt

The final way to change one of Autolog's special keys is to enter the name of the key as a command *without an argument.* You will be prompted to type the character you want to assign to that key. You can enter control characters by holding down the "control" key while you press the key.

☞ You may not enter the following characters directly using this method. You must use the ASCII decimal equivalent as an argument to redefine a special key as one of these characters:

carriage return	delete	rubout
cancel key	linefeed	space

Also, the command **`break`** without an argument will actually send a break condition to the remote system while in command mode. You must use one of the other two methods described earlier to redefine the **break key**.

2.7 The `carrier` Command

The `carrier` command causes Autolog to monitor the **carrier detect** hardware signal (also called DCD) and to exit talk mode and return to command mode if carrier is lost. If a file transfer was in progress, it will be aborted. Enter

`carrier` *true or false*

to turn the carrier detect option on or off.

Your communications port interface hardware and software must be capable of supporting and reporting the DCD signal for this option to work. Refer to the ***Autolog Installation and Platform Guide*** for more information.

2.8 The `control` Command

The `control` command causes all characters, even control and other nonprintable characters, to be displayed on your screen in a printable format. The `control` option also displays hardware control signal changes and serial communication errors if they occur. It can be a useful diagnostic tool. Enter

`control` *true or false*

to turn the `control` option on or off.

When the `control` option is turned on, nonprintable characters received from the remote system will be displayed in the following formats:

- Control characters will be preceded by the **meta** character. For instance, a carriage return (control-M) will appear as *`^M`*.
- Non-ASCII characters (whose decimal values are greater than 127) will be preceded by the % character. For instance, a lowercase “b” with the high bit on would appear as *`%b`*. A control-G with the high bit on would appear as *`%^G`*.

- *DEL* will be displayed for the delete character (ASCII decimal value 127). *ESC* will be displayed for the escape character (^[, ASCII decimal value 27).

☞ If incoming characters are currently being written to a file using the `get` command (see Section 3.6 in Chapter 3), nonprintable characters will be handled normally (i.e., written to the file) even when `control` is active.

2.9 The `debug` Command

The `debug` command is another useful diagnostic tool that displays various types of diagnostic information. Enter

`debug` *setting true or false*

to turn on or off the various diagnostic displays indicated by *setting*. The valid settings are **`crt`**, **`input`**, **`output`**, **`port`**, **`protocol`**, **`script`**, and **`uart`**. These settings indicate the various types of information that can be displayed, as discussed in the following sections.

`debug crt`

This setting causes screen formatting and cursor positioning characters to be displayed in a printable format on your screen. In addition, the screen commands and also other nonprintable characters written to a `get` file (see Section 3.6 in Chapter 3) while `debug crt` is active will also be written in a printable format.

Screen formatting and cursor positioning commands will be written in a special format. Cursor positioning commands will appear in this format:

<row,col>

where *row* and *col* are numbers indicating the row and column where the cursor is to be positioned. For instance, *<2,3>* would indicate a command to send the cursor to row 2, column 3 of the screen. Screen formatting commands (such as "erase to end of screen") will appear in this format:

<−num,num>

The first number will be negative to distinguish screen formatting from cursor positioning commands. Appendix B lists the screen formatting commands that correspond to the numeric codes used by the `debug crt` option.

`debug input`

This setting will display characters received from your modem that are normally not displayed. For instance, the "connect" message that your

modem generates when it establishes a connection after dialing the remote modem, which is normally suppressed, will be displayed on the screen when the `debug input` option is turned on. `debug input` also displays characters receive by the Autolog XCALL supported on some platforms.

debug output

This setting will display the commands Autolog sends to your modem, which are normally not displayed. For example, the "atd" command to dial the modem, which is normally suppressed, will be displayed on the screen when the `debug output` option is turned on. `debug output` also displays characters sent by the `say` command (see Section 4.5 Chapter 4) and by the Autolog XCALL supported on some platforms.

debug port

The `debug port` option works only on UNIX and AIX systems.

The `debug port` option displays diagnostic information about your serial communication port whenever you enter a command that affects the port (e.g., a `link`, `baud`, or `parity` command—see Section 1.3 in Chapter 1). What you see on your screen will resemble Figure 2.2. The device communications settings displayed will be helpful to Soft Machines' technical support staff if you need to call for help diagnosing a problem.

```
>link tty1A
ttylocking port
pid_str=       807

opening port
CBAUD bits=13
c_iflag=00, c_oflag=00, c_cflag=06275
c_lflag=00, c_line=00
c_cc[0-7]={127 28 8 21 0 0 255 255 }
c_iflag=00, c_oflag=00, c_cflag=06275
c_lflag=00, c_line=00
c_cc[0-7]={127 28 8 21 0 0 255 255 }
>
```

Figure 2.2 `debug port` display.

debug protocol

The `debug protocol` option displays diagnostic information about error-correcting file transfers. See Chapter 3 for more information about file transfers and the `debug protocol` command.

debug script

The `debug script` command enables single-stepping through a script file. See Chapter 4 for more information about script files and the `debug script` command.

debug uart

The `debug uart` setting displays serial communication errors that your computer's hardware and software are capable of detecting and reporting. Errors that occur while in talk mode are displayed. Changes in modem control hardware signals and breaks detected while in talk mode are also displayed.

The `debug uart` option is not available on UNIX and AIX systems.

Figure 2.3 illustrates what you might see while in talk mode while `debug uart` is active. Signals in lowercase are inactive, signals in uppercase are active. When the remote modem answers after dialing its number (shown in the figure as an atdt command), the **carrier detect** (DCD) signal becomes active. This change in a modem control signal triggers the `debug uart` display. The CTS (clear to send), DSR (data set ready), and DCD (data carrier detect) signals are active, while the ri signal (ring) is inactive.

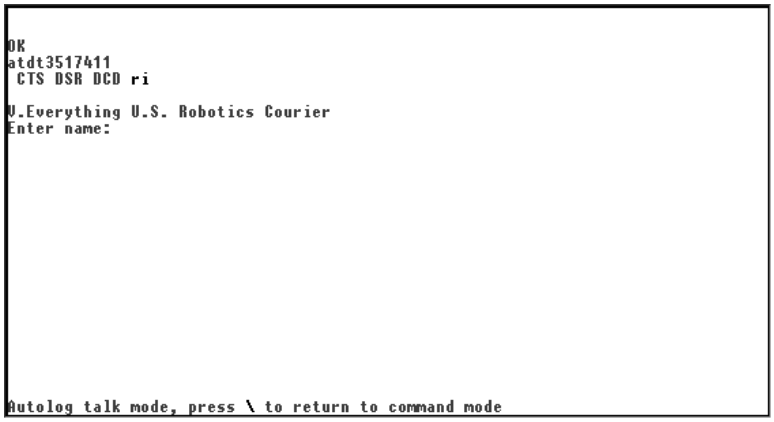


Figure 2.3 `debug uart` summary display.

These are the types of serial communication errors that can be reported by `debug uart`:

Framing errors. Framing errors indicate that a received character had no stop bit. This can be a symptom of characters damaged by line noise or of mismatched baud rates.

Overrun errors. Overrun errors indicate that incoming characters are not being read quickly enough by your computer. This problem can be minimized by using hardware flow control (see the discussion of the `flow` command in Section 1.3 in Chapter 1), but it may not be possible to eliminate this problem altogether. If you experience many overrun errors, you may need to lower the baud rate at which you connect with the remote system.

Character parity errors. A parity error indicates that a character was received with the wrong parity. This can be a sign that you need to use the `parity` command (see Section 1.3 in Chapter 1), or it can be a symptom of characters damaged by line noise.

Breaks detected. The number of break conditions received will be displayed. Breaks detected may be legitimate breaks sent by the remote system or may indicate a mismatch in baud rates.

Modem flow control changes. The number of times your modem signaled to its communications port to start or stop sending data using hardware flow control. Many modem flow control changes indicate that a slow or busy local computer system or a noisy phone line is keeping your modem very busy, and this could be slowing down your file transfers. If you're experiencing trouble with file transfers but no modem flow control changes are indicated, your hardware may not be configured to support hardware flow control. Refer to the ***Autolog Installation and Platform Guide*** to see if hardware flow control, which will improve file transfer performance, is available for your computer system.

2.10 The `guard` Command

The `guard` command discards nonprintable characters before they are displayed on the screen and before they are recorded in a disk file using `get` (see Chapter 3). Some nonprintable characters may cause your terminal to perform undesirable functions (such as turning off scrolling or locking up the keyboard so you can't type). Nonprintable characters may also make disk files created with `get` unreadable by your text editor or word processing software. Enter

`guard` *true or false*

to turn the `guard` option on or off.

The `guard` option will allow all printable ASCII characters and the following control characters to be displayed on your terminal or written to a disk file with `get`:

backspace (control-H)	formfeed (control-L)
bell (control-G)	linefeed (control-J)
carriage return (control-M)	tab (control-I)

The `guard` option will prevent the displays of screen-oriented software on the remote system from being displayed correctly on your screen, because it will filter out the characters used to format the screen.

2.11 The `terse` Command

The `terse` command turns off Autolog's full, formatted command screen. Terse command mode is indicated merely by Autolog's command prompt, >. Use the command

`terse` *true or false*

to activate or deactivate terse command mode. Terse mode is useful in script files (see Chapter 4) or whenever you don't want the screen "repainted" when you switch from command mode to talk mode.

Chapter 3
File Transfers

Transferring files between computer systems may be one of the things you do most frequently with Autolog. In order to transfer a file with a remote computer, you must use a **file transfer protocol** that the remote computer supports. There are two broad classes of file transfer protocols: those that support **error correction** and those referred to as "no protocol," ASCII, or text file transfers that don't support error correction. You will usually want to use an error-correcting protocol if the remote system supports one. However, you can use Autolog's text file transfer commands to exchange files with any type of computer, even if a more sophisticated protocol is unavailable.

In this chapter, we'll first look at error-correcting file transfer commands. Sections 3.6 and 3.7 at the end of this chapter will explain Autolog's "no protocol" text file transfer commands. We will use **upload** and **download** to indicate the direction of file transfers: *uploading* is sending a copy of a local file *to* the remote system; *downloading* is receiving a copy of a file *from* the remote system to your local system.

In general, any file transfer in progress can be aborted by pressing the **cancel key**. Where exceptions exist, these are discussed in the appropriate sections.

3.1 SMT Protocol: `transmit` and `receive`

Autolog supports a proprietary file transfer protocol, SMT, that can be used to exchange files with computers that are equipped with Autolog's

Slave program. Slave is supported by UNIX and AIX, DOS/Windows, and AMOS computers; you may provide remote systems with which you transfer files with a copy of the Slave utility so that you can use Autolog's `receive` and `transmit` commands to easily exchange files with those system. Instructions for installing Slave on remote systems are found in the ***Autolog Installation and Platform Guide.***

SMT protocol supports error detection and correction, data compression, and buffering, making it one of the fastest and most reliable file transfer protocols. It is also frequently the easiest to use, since typically no advance preparation of the remote system is necessary before beginning a file transfer, once the Slave program has been installed. (On remote DOS and Windows systems, the Slave program must already be running before you call in.)

Uploading Files with `transmit`

If the remote system has a copy of Slave, you can send it a copy of a file on your system with the `transmit` command. Enter the command

`transmit` *file(s)*

where *file(s)* is the specification(s) of the file(s) you wish to send. The file specification may be a single file, may contain wildcard symbols to send a group of files, or may consist of a list of files (with or without wildcard symbols). For example,

```
transmit myfile.lit
transmit my*.exe
transmit new* old* myfile.txt
```

are all acceptable `transmit` commands.

If no account or directory is indicated, Autolog will look for files only in the current local directory. You may indicate a different directory in the file specification, according to the conventions honored by your operating system.

`transmit sys:*.lit` (AMOS)
`transmit *.lit[1,4]` (AMOS)
`transmit usr/bin/myfile` (UNIX or AIX)
`transmit ../[a-c]*` (UNIX or AIX)
`transmit c:\usr\myfile.exe` (DOS/Windows)
`transmit ..\my*` (DOS/Windows)

☞ If you give only one file specification to be used for both the local and remote systems, and that specification includes a path or directory as in the preceding examples, the same path or directory will be used on the remote system as well. This means the remote system must also have the same path or directory in which to place the files, or your file transfer will fail. However, Ashell users can

transfer files to AMOS systems from analogous Ashell directories. For example, an Ashell user can enter **`transmit dsk13\001006\myfile.txt`** to transfer the file myfile.txt to the AMOS directory DSK13:[1,6].

The use of wildcard symbols, whether commas or spaces separate lists of file specifications, and the way to indicate directories or accounts depend on your operating system.

If you are transferring files to a computer that uses a different operating system or different file-naming conventions, Autolog will automatically convert the file name to a style appropriate for the remote system if necessary. For instance, if you transmit a file with 8 characters in its name to a system that only accepts 6-character file names, the file name will be truncated. Characters that are illegal in file names on the remote system will be omitted (e.g., my_file.txt on a DOS/Windows computer would become myfile.txt when transferred to an AMOS computer).

You may give files new names as they are transferred by using the syntax

`transmit` *new* **`=`** *old*

where *new* is the name you want to give the new copy of the file on the remote system and *old* is the name of the existing file on your system. You can also use wildcarding when renaming files, as in

`transmit *.new = *.old`

The *new* file specification must be a valid file specification on your local system. Otherwise, you must use the "==" syntax described next.

If you want to disable Autolog's automatic file name conversion or to use a new (remote) file specification that is not legal on your local system, use two equal signs, as in the command

`transmit` *new* **`==`** *old*

The *new* file specification must conform to any file name restrictions of the remote operating system.

You cannot use wildcard symbols or transfer more than one file at a time using "==" syntax.

When the file transfer begins, Autolog will display the progress with a bar graph, as illustrated in Figure 3.1. If `terse` mode is active (see Chapter 2, Section 2.11) an abbreviated, text-only report will be displayed.

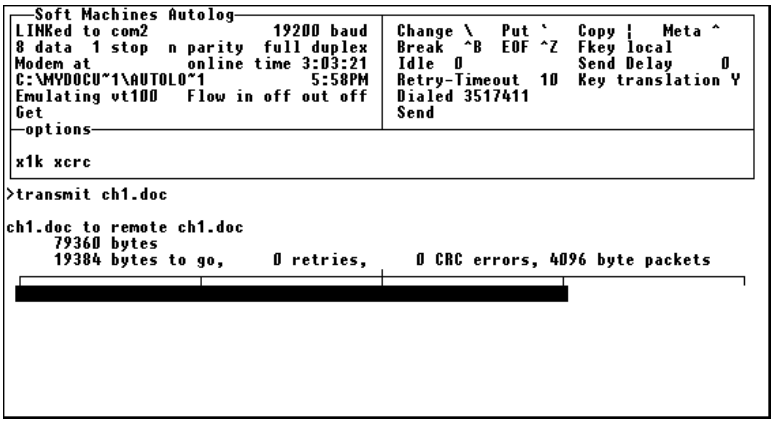


Figure 3.1 A `transmit` file transfer in progress.

After the file transfer is complete, Autolog will display a report that includes the time it took to transfer the file and the effective baud rate (a measure of how efficiently the file was transferred indicating how many characters per second were transferred), as illustrated in Figure 3.2.

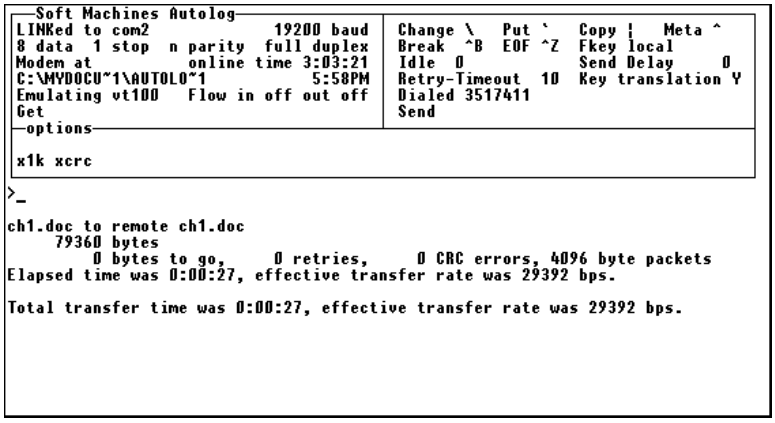


Figure 3.2 A completed `transmit` file transfer.

Downloading Files with `receive`

If the remote system can run Slave, you can get a copy of a remote file to your system with the `receive` command. Enter the command

`receive` *file(s)*

where *file(s)* is the specification(s) of the file(s) you wish to send. The file specification may be a single file, may contain wildcard symbols to send a group of files, or may consist of a list of files (with or without wildcard

symbols). See the preceding section on the `transmit` command for examples of valid file specifications.

If no account or directory is indicated, Autolog will look for files only in the current directory on the remote system. You may indicate a different directory in the file specification, according to the conventions honored by your operating system.

☞ If you give only one file specification to be used for both the local and remote systems, and that specification includes a path or directory, the same path or directory will be used on your local system as well. This means your local system must have the same path or directory in which to place the files as the one you specify, or your file transfer will fail. However, Ashell users can receive files from AMOS systems to analogous Ashell directories. For example, an Ashell user can enter **`receive dsk13:myfile.txt[1,6]`** to transfer the AMOS system's file myfile.txt to the Ashell directory dsk13\001006\.

 The use of wildcard symbols, whether commas or spaces separate lists of file specifications, and the way to indicate directories or accounts depend on your operating system.

Autolog will automatically convert the file name to a style appropriate for the remote system if necessary. You may give files new names as they are transferred by using the syntax

`receive` *new* **`=`** *old*

where *new* is the name you want to give the new copy of the file on the remote system and *old* is the name of the existing file on your system. You can use wildcard symbols to rename batches of files using this syntax:

```
receive *.new = *.old
```

☞ The *old* file specification (the name of the file on the remote system) must conform to any file name restrictions on your local operating system. If the existing (remote) file's specification is not legal on your system, you must use the "==" syntax described next.

If you want to disable Autolog's automatic file name conversion or to specify remote files whose specifications are not legal on your local system, use two equal signs, as in the command

`receive` *new* **`==`** *old*

☞ *You cannot use wildcard symbols or transfer more than one file at a time* using "==" syntax.

The `receive` command, like the `transmit` command, will display the progress and completion of the file transfer, similar to the displays in Figures 3.1 and 3.2.

Switches for SMT Protocol

There are a number of switches that can be used to select files to be transferred using SMT protocol. They can be used singly or in conjunction along with wildcard symbols to select exactly the files you want to transfer.

The behavior and format of file transfer switches depend on your operating system. Not all switches are honored by all operating systems. See the ***Autolog Installation and Platform Guide*** for details on which switches your computer supports and additional information on their placement in a `transmit` or `receive` command. If the switch is placed immediately after the command, with no intervening spaces, the **/** "switch" character will work on any platform, for example, **`transmit/h`** or **`receive/q`**. You can also precede a switch with the usual "switch" character, either **/** or **–**, for your operating system. For DOS, Windows, UNIX, and AIX computers, the switch(es) should come immediately after the command, before any arguments, for example, **`transmit/v my*.exe`** or **`transmit -q my*`**. For AMOS computers, the placement of the switch(es) determines whether the switch is local (applies to a single argument), global (applies to the entire command), or "sticky" (applies to following but not preceding arguments). Any switch can be spelled out in its entirety or shortened to the fewest number of letters that are unique, for example, you can use either **`q`** or **`query`** for the query switch.

q (Query) Switch

The `q` (query) switch lets you verify each file before it is transferred. Autolog prompts you to choose whether to transfer each file in a batch of files. Answer **`Y`** to transfer that single file or **`N`** to skip that file. A typical batch file transfer might look like the one in Figure 3.3.

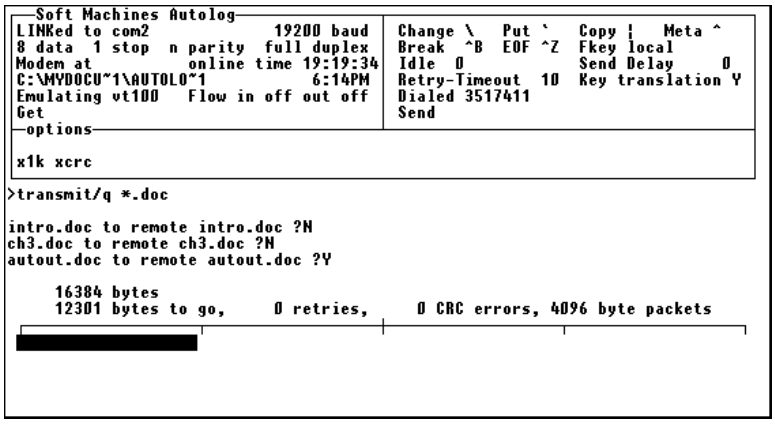


Figure 3.3 Using the `q` (query) switch.

`h` (Hash) Switch

The `h` (hash) switch allows you to transfer only those files that have a different **hash code** between systems, indicating that their contents are not identical. If a file of the same name as the original does not already exist on the destination system, the file will always be transferred.

`nod` (Nodelete) Switch

The `nod` (nodelete) switch prevents overwriting existing files on the destination system. Normally, unless the `noerase` option is turned on (discussed a little later in this chapter), Autolog will replace existing files on the destination with new files from the originating system if they have the same name. You can use the `noerase` option discussed later to prevent such overwriting for all your file transfers, or you can use the `nod` switch to prevent overwriting in a single file transfer command.

`t` (Text) Switch

The `t` (text) switch converts line endings in text files to the appropriate type of line ending for the destination system. For example, use the `t` switch when transferring a file from an operating system that uses "control-M control-J" for its line endings to one that uses "control-J" alone for line endings. You'll know if the destination system uses a different type of line ending from the source system if text files that you transfer appear doublespaced or appear to have all the text in one giant line.

Date and Time Switches

You can use Autolog's date and time switches to select files to transfer based on their creation (the `cdate` switch), last modification (`mdate`), or last backup (`bdate`) dates or times. You can select only files that exactly match the specified date or time, or you can use the `before` and `after` switches to select files created, modified, or backed up on or before a given date or time, or on or after the specified date or time.

Dates and times should be entered in this format:

mm-dd-yy@hh:mmAM

where *mm-dd-yy* is the date in numeric, month-day-year format (e.g., January 30, 1996 would be **`01-30-96`**) and *hh:mmAM* is the 12-hour time with the indication `AM` or `PM` (e.g., **`2:00PM`**). By convention, `12:00AM` indicates midnight, `12:00PM` indicates noon, and midnight of a given day is considered the beginning of the day (e.g., **`01-30-96@12:00AM`** indicates the midnight between January 29 and January 30).

The desired date and time switch should be followed by a colon, then the desired date, time, or both, as in these examples:

cdate:02-07-96@2:00PM (files created at exactly 2 PM on February 7, 1996)

bdate:07-21-96/before (files backed up at any time on or before July 21, 1996)

mdate:@9:32AM -after (files modified on any day at 9:32 AM or any time until midnight thereafter)

For comparison purposes, a blank date and time field is considered to be before any actual date or time.

v (Version) Switch

The `v` (version) switch can be used to transfer only those files that have a different version number on the destination system than they do on the originating system. If the file does not already exist on the destination system, it will always be transferred. If the original file does not have a system-identifiable version number (such as a text file), the file will always be transferred. However, if a file of the same name already exists on the destination system and it has the same version number as the original file, that file will *not* be transferred.

The version switch works only when comparing AMOS files that have AMOS-style version numbers.

File Type Switches: `contig`, `seq`, and `randomize`

Under the AMOS operating system, which distinguishes between contiguous (random-access) and sequential files, you may use the `/contig` or `/seq` switch to select only files of the specified type.

The `randomize` switch can be used with the `transmit` command to create a random file when sending a file *to* an AMOS system. Usually, when sending files from a system that does not distinguish between contiguous and sequential files, the files transmitted to an AMOS system are created as sequential files. By adding the `randomize` switch to your `transmit` command, however, the files you transfer will be created as contiguous (random-access) files. The `randomize` switch is useful for transferring data files or other files that AMOS applications need as contiguous files..

SMT Protocol Options

There are a number of options that can be activated to control how the `transmit` and `receive` commands behave. You activate these options by entering a command before you begin a file transfer. The option stays activated until you turn it off again, so it can be active during multiple file transfers or during an entire communications session.

`alarm` Option

When the `alarm` option is active, a "beep" or "bell" will sound when an entire file transfer command is completed. Enter the command

`alarm` *true or false*

to turn the alarm on or off.

The `alarm` option can be handy when transferring a large batch of files if you want to be alerted when the entire batch has been transferred. The alarm normally will sound only after all the files requested by a single file transfer command have been transferred. However, if you use the `q` (query) switch, the alarm will sound for each file in the batch when Autolog asks you if you want to send that file.

☞ The `alarm` option will work with any of Autolog's error-correcting file transfer commands.

`nocompress` Option

The `nocompress` option deactivates the automatic data compression of the SMT protocol. Autolog's SMT protocol normally compresses repeated characters to make file transfers more efficient. Enter the command

`nocompress` *true or false*

to deactivate data compression (**`true`**) or turn data compression back on (**`false`**).

`noerase` Option

The `noerase` option prevents accidentally overwriting existing files when transferring like-named files. Enter the command

`noerase` *true or false*

to turn the `noerase` option on or off. While the `noerase` option is active (the word *`noerase`* will appear in the "options" box of the Autolog command screen), if you try to transfer a file that already exists on the destination system, you will receive the message *`file already exists`* or *`REMOTE file already exists`*, and the file will not be transferred. When transferring a batch of files using wildcarding or multiple file specifications, only files that already exist on the destination system will be skipped. New files will transfer normally.

☞ The `noerase` option can be used with `transmit`, `receive`, or when *downloading* files with other file transfer protocols to prevent *local* files from being overwritten. It may also prevent remote files from being overwritten with some remote ZMODEM programs, although it is not advisable to rely on it working with `ztransmit`.

Adjusting the Packet Size with `packetsize`

The `packetsize` command allows you to select the size of each "chunk," or **packet**, of data that is sent in a file transfer. To select a packet size, enter the command

`packetsize` *number*

where *number* is the new initial packet size (in bytes) that you want to use. Valid packet sizes range from 64 to 4096 bytes. The normal, default packet size is selected using the Autfix program (see the ***Autolog Installation and Platform Guide***) but varies depending on the packet size supported by the remote computer.

The `packetsize` command normally adjusts the size of the *initial* packet sent during a file transfer. Autolog dynamically adjusts the packet size as the file transfer progresses to optimize the file transfer speed. If you don't want Autolog to change the packet size (i.e., you want to use a fixed packet size), use the `/f` switch:

`packetsize` *number***`/f`**

Selecting a fixed packet size will force *all* packets sent, not just the initial packet, to be that size. Note that, regardless of the "switch character" normally used on your system, the `/f` switch for `packetsize` must always use the slash character.

☞ Very old AMOS versions of the Slave program may not support an adjustable packet size. The packet size will also depend on the amount of memory available for Autolog's and Slave's use on the local and remote computers. The packet size may be automatically adjusted downward if enough memory is not available for the packet size you select.

☞ If you or the remote system is using a pseudo-duplex modem (a modem that *simulates* full duplex, such as the DataRace Race I, Race II, or BMX), we strongly recommend that you use the `packetsize` command with the `/f` switch to select as large a fixed packet size as possible to speed up the transfer and reduce the number of retries.

`timeout` Option

The `timeout` option adjusts the amount of time Autolog will wait before attempting to retransfer a **packet** or acknowledgment. The `timeout` command has the format:

`timeout` *number*

where *number* is the number of seconds you want Autolog to wait before attempting a retransmission. The default value is selected using the Autfix program (see the ***Autolog Installation and Platform Guide***). The current value is displayed in the upper right section of the Autolog command

screen as the *Retry-Timeout* value. Autolog monitors the length of time it takes for packets to be acknowledged and may allow extra time if conditions warrant.

The default timeout value of 10 seconds was selected to be as short as possible while still taking possible turn-around delays (from busy systems or from satellite links) into account to maximize file transfer efficiency. When adjusting the timeout value, be sure to consider such possible delays. A timeout value that is too small will result in unnecessary retransmissions. Remember that Autolog will wait the maximum timeout period only if a packet or acknowledgment receives no response from the remote system.

xnet Option

The xnet option helps SMT file transfers work over X.25 and Telnet type connections. The xnet option encodes certain characters that prevent file transfers from working over these type of connections. Turning on the xnet option therefore makes file transfers less efficient, but it is often necessary to make file transfers work over X.25 and Telnet connections.

Use the command

xnet *true or false*

to turn the xnet option on or off.

7-Data-Bit File Transfers

Most error-correcting file transfer protocols, including Autolog's SMT protocol, normally require an 8-bit communications path. This is because, although the file to be transferred may contain only 7-bit ASCII text characters, 8-bit characters are used for control functions like error correction and data compression. Although SMT protocol works most efficiently with an 8-bit data path, it will work with a 7-bit path too. To use 7-bit SMT file transfers, you only need to configure your communications channel with 7 data bits and parity, using the commands **data 7** and **parity even** or **parity odd** (as explained in Chapter 1).

☞ A problem frequently encountered with error-correcting file transfers is that, unbeknownst to you, the remote modem (or sometimes your own local modem) is configured to strip off the eighth bit of characters it passes between your computer and the remote system. This can prevent error-correcting file transfers from working. If your SMT file transfers consistently fail when using a particular modem or when calling a certain remote system, try selecting **data 7** and **parity even** or **parity odd** when calling that system or with that modem.

Troubleshooting SMT File Transfers

Most of the standard file errors that you might receive on your system or on the remote system when manipulating files (such as "permission violation," "device write protected," and so on) can also occur when trying to create or overwrite files using a file transfer command. Autolog will report such file error messages if they occur, preceded by the word *REMOTE* if the error occurred on the remote system, or as they normally appear on the local computer if a local error.

The error *file already exists* occurs when using the noerase option or nod (nodelete) switch if you attempt to transfer a file that already exists on the destination system.

The error *transfer retry count exceeded* occurs after Autolog has tried to transfer a packet the maximum number of times without success and so has aborted the file transfer. The maximum number of attempts is 12, unless the Autfix program has been used to change this default (see the ***Autolog Installation and Platform Guide***).

SMT file transfer protocol requires that both the local and remote modems pass all characters transparently. Modems that are configured to use software flow control (which requires that special "in-band" characters such as XON/XOFF or ENQ/ACK be reserved for flow control) or modems that are configured to discard rather than pass certain characters will not work properly with SMT 8-bit file transfers. When either the local or remote modem does not pass all characters, transmit and receive file transfers may appear to get "stuck," may consistently fail with the error *transfer retry count exceeded*, or may result in *REMOTE system does not respond* or *modem does not respond* errors. The solution to this problem is to use 7-data-bit file transfers (discussed above), which will also encode problematic flow control characters. The same symptoms may appear when transferring files over X.25 or Telnet-type connections, which also can't allow the transparent transfer of all 8-bit characters. The xnet option can be used to encode the problematic characters when using these types of connections.

Modems that use nonstandard character sequences to hang up (such as the Racal Vadic Maxwell modem, which can be configured to hang up when it receives the sequence control-C control-D) may hang up in the middle of a file transfer. Certain files simply cannot be transferred using such modems unless they can be reconfigured to avoid hanging up on such nonstandard sequences.

The error *REMOTE system does not respond* occurs when Autolog cannot start up the Slave program on the remote system to perform a file transfer. Sometimes simply reentering the file transfer command will clear up this problem. You may also receive the error message *?Cannot execute Slave on remote system*. If a

second attempt also fails to start the Slave program, you need to check that Slave has been properly installed on the remote system and that there are enough system resources available to use it. Press the **change key** to enter talk mode, then type the command **slave** **Enter** at the system prompt. You should see a display similar to the one in Figure 3.4 if Slave is installed and usable.

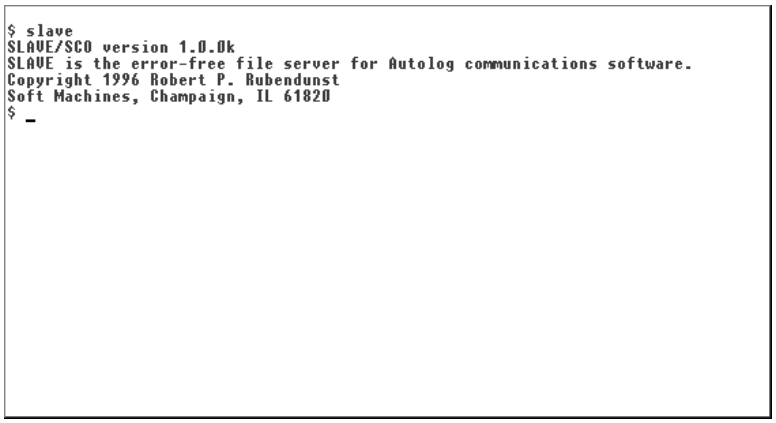


Figure 3.4 Entering the Slave command manually.

If you receive an error message or an indication that the Slave program does not exist, refer to the ***Autolog Installation and Platform Guide*** for help installing Slave.

If you have a problem that you can't resolve on your own, Soft Machines' technical support can gather useful diagnostic information from the `debug protocol` command. Turn this option on with the command:

```
debug protocol true
```

and try your file transfer again. The `debug protocol` option causes diagnostic information about the file transfer to be displayed on your screen as the file transfer progresses. The information will vary depending on the file transfer protocol in use, but it may resemble the display in Figure 3.5.

```
Outgoing check=167215 for 9 bytes
Packet ACK #1 sent
Receiving karen.txt as KAREN.TXTconvert_error(2(No such file or directory)

Out-queuing FREPLY sequence # 2
Outgoing check=101474 for 93 bytes
Packet FREPLY #2 sent
!Recv'd packet type: ACK, size: 13, Seq:2
Dequeuing block FREPLY #2 from repacket
!Recv'd packet type: DAT, size: 45, Seq:2
Queing ACK #2
Out-queuing ACK sequence # 2
Outgoing check=125215 for 9 bytes
Out-queuing FILACK sequence # 3
Outgoing check=113143 for 9 bytes
Packet ACK #2 sent
Packet FILACK #3 sent
!Recv'd packet type: ACK, size: 13, Seq:3
Dequeuing block FILACK #3 from repacket

Elapsed time was 0:00:01, effective transfer rate was 320 bps.

Total transfer time was 0:00:01, effective transfer rate was 320 bps.
>_
```

Figure 3.5 `debug protocol` display.

Manually Releasing Slave

When you enter a `transmit` or `receive` command, Autolog automatically starts up the remote Slave program. The Slave program runs until you press the **change key** or until you enter a `hangup`, `link`, `unlink`, `finish`, or `say` command or another type of error-correcting file transfer command (e.g., `xreceive` or `ztransmit`), at which time Autolog will automatically stop the Slave program. However, if an accident (such as an unforeseen disconnection of the phone line or a power outage) causes Autolog to lose track of whether the Slave program is running on the remote system, you may need to manually terminate the Slave program using the **`release`** command.

You can tell whether the Slave program is still running if you are still connected (or reconnect) to the same remote port on the remote system that you were using originally for file transfers. The port will appear "dead," that is, it will not echo characters you type or respond to commands you give it. In this situation, press the **change key** to return to command mode and type the command **`release`**. Autolog will then stop the Slave program.

In the event that a *remote* Autolog user calls *into* your local system and leaves a local port running Slave, you can generally use a "kill process" command such as the Kilslv program (for AMOS) or `kill` (for UNIX and AIX) to terminate Slave. Consult the ***Autolog Installation and Platform Guide*** for more details.

3.2 ZMODEM File Transfers

Autolog supports the family of X-, Y-, and ZMODEM file transfers, including variations such as X- and YMODEM-g "turbo" transfers. The

first of this family of protocols that we will discuss is ZMODEM. ZMODEM is supported by many different types of computer systems and is the most sophisticated of the X-, Y-, ZMODEM family, making it a very popular file transfer protocol

You can transfer a file using ZMODEM with the `ztransmit` and `zreceive` commands. You will usually need to prepare the remote system for a ZMODEM file transfer while in talk mode, then press the **change key** to enter command mode before issuing a `ztransmit` or `zreceive` command.

Uploading Files with `ztransmit`

The `ztransmit` command is used to send a file or batch of files to the remote system using ZMODEM protocol. Usually, you will need to prepare the remote system while in talk mode to begin receiving a file using ZMODEM before using the `ztransmit` command. However, if the remote system supports the `rz` command (which instructs it to begin receiving files with ZMODEM protocol), no advance preparation is necessary.

These are the steps to do a `ztransmit` upload:

1. If the remote system supports the `rz` command, skip to step 2. Otherwise, while in talk mode, prepare the remote system to receive a file using ZMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that files will be sent (the remote system will be *receiving* the file), or tell it to use ZMODEM rather than another file transfer protocol.
2. If you're in talk mode, press the **change key** to enter command mode. Then enter the command

   `ztransmit` *file(s)*

   where *file(s)* are the file specification(s) of the file(s) you want to send, which may contain wildcard symbols. Other formats and options for the `ztransmit` command are available and are discussed below.

As the file transfer progresses, a bar graph similar to the one in Figure 3.6 (or an abbreviated, text-only report if the `terse` option—discussed in Chapter 2—is turned on) will be displayed.

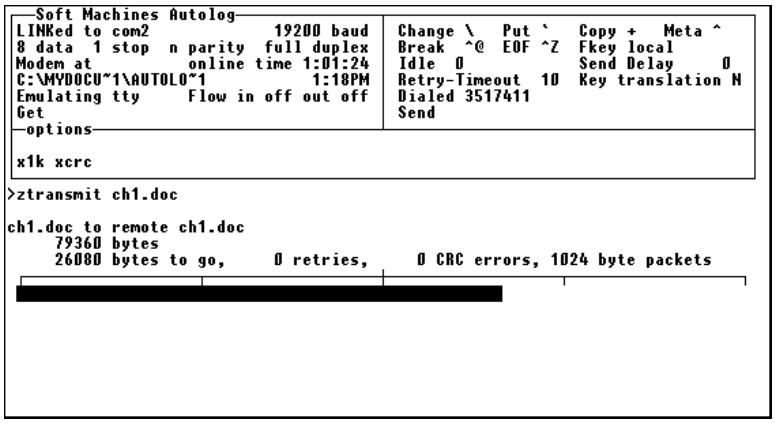


Figure 3.6 `ztransmit` file transfer in progress.

You can specify a single file or use wildcarding symbols (according to the wildcarding conventions supported by your local system) to specify a batch of files to send, as in these examples:

```
ztransmit myfile.lit
ztransmit ??file
ztransmit myfile[1-20]
```

You may rename files as they are transferred:

`ztransmit` *new* **=** *old*

☞ The *new* (remote) file specification must be a valid file specification on your local system as well. Otherwise, you must use the "**==**" syntax described below.

You can use wildcarding symbols and also any of the file transfer switches (discussed in the section "Switches for ZMODEM Protocol" below) with the formats of the `ztransmit` command discussed so far. When transferring files between unlike operating systems, Autolog will automatically convert file names that are not legal on the remote system to names that are acceptable by the receiving system. If you want to disable Autolog's automatic file name conversion or to use a new (remote) file specification that is not legal on your local system, use two equal signs:

`ztransmit` *new* **==** *old*

where *old* is the local file specification and *new* is the file specification for the remote system. Using the "**==**" syntax, you can transfer only one file at a time, and you cannot use any file transfer switches or wildcard symbols.

Downloading Files with `zreceive`

The `zreceive` command is used to receive a file or batch of files from the remote system using ZMODEM protocol. You must prepare the remote system while in talk mode to begin sending a file or files using ZMODEM before using the `zreceive` command.

These are the steps to perform a `zreceive` download:

1. While in talk mode, prepare the remote system to send a file or files using ZMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that files will be sent (the remote system will be *sending* the file), or tell it to use ZMODEM rather than another file transfer protocol. There are many variants of ZMODEM: Autolog's `zreceive` command will work with many of them. If the remote system offers several variants, you may need to experiment to discover which variant works most efficiently. The actions you need to begin the transfer and the way to specify files depend on the remote system's ZMODEM software. If the `zauto` option is turned on (discussed in the next section), this is all you need to do. If `zauto` is not turned on, proceed to step 2.
2. Press the **change key** to enter command mode. Then enter the command

   `zreceive`

   Notice that no file specification is needed in the `zreceive` command: The file specifications, according to the ZMODEM protocol convention, are determined by the sending (remote, in this case) system. You may, however, rename files as they are received or use file transfer switches as discussed below.

As the file transfer progresses, a bar graph similar to the one in Figure 3.6 (or an abbreviated, text-only report if the `terse` option—discussed in Chapter 2—is turned on) will be displayed.

If you want to rename the file(s) as they are received, use this format of the `zreceive` command:

`zreceive` *new* **`= *`**

where *new* is the new file specification(s) (e.g., a single file specification or, for a batch of files, a wildcard specification such as **`*.new`**). Notice that the file specification to the right of the equal sign should always be **`*`** or **`*.*`**, because the existing, remote files are specified on the remote system when starting the file transfer and don't need to be specified again in the `zreceive` command.

`zauto` Option

The `zauto` option allows you to receive files using ZMODEM protocol without using the `zreceive` command. While the `zauto` option is turned on, Autolog will automatically begin receiving files as soon as the remote system begins sending them using ZMODEM protocol (usually while you are still in talk mode). A text-only report of the file transfer progress will be displayed.

Enter the command

`zauto` *true or false*

to turn the `zauto` option on or off.

☞ If you want to rename files as they are received or to use any of the file transfer switches, you must turn off `zauto` and use the `zreceive` command in the appropriate format. Also, be sure to turn off the `zauto` option before attempting file transfers using a different protocol.

Switches for ZMODEM Protocol

A wealth of file transfer switches is available to select files and control how ZMODEM file transfers behave. File transfer switches can be shortened to the fewest number of unique letters (e.g., `nostamp` can be abbreviated as `nos`).

The behavior and format of file transfer switches depend on your operating system. See the ***Autolog Installation and Platform Guide*** for additional information on their placement in a `ztransmit` or `zreceive` command. If the switch is placed immediately after the command, with no intervening spaces, the **`/`** "switch" character will work on any platform, for example, **`ztransmit/h`** or **`zreceive/q`**. You can also precede a switch with the usual "switch" character, either **`/`** or **`–`**, for your operating system. For DOS, Windows, UNIX, and AIX computers, the switch(es) should come immediately after the command, before any arguments, for example, **`ztransmit/newer my*.exe`** or **`zreceive –q my*`**.

Many switches allow comparisons of file dates and times. For operating systems that store more than one data-and-time stamp for files, the date and time used will be the update (or last modified) date-and-time. Refer to the ***Autolog Installation and Platform Guide*** for details on the handling of file dates and times.

Using a number of ZMODEM file transfer switches at once can be tricky. First, you must make sure that the options you request don't directly conflict each other (e.g., using the `existing` and `nodelete` switches together would result in no files being transferred, because the former selects only files that already exist on the destination system but the latter prevents existing files from being overwritten). Second, if the

options you select using Autolog's switches are in conflict with options selected by the remote system's ZMODEM software, the results are unpredictable. Third, if options are selected that are not supported by the remote system, the results are also often unpredictable. (According to the protocol, if an option is not supported, it should be ignored, and the file transfer should proceed as if the switch or option were not there. However, our experience informs us that not all ZMODEM programs behave predictably in this case.)

append

The `append` switch causes the file being transferred to be appended to the end of the destination file, if it already exists. If the file doesn't already exist on the destination system, the file will be transferred as normal (i.e., a new file will be created). `append` cannot be used in conjunction with the `different`, `hash`, `longernewer`, `newer`, or `nodelete` switches.

different

The `different` switch allows the transfer of only files that either don't already exist on the destination system or that differ in size or in date and time. `different` cannot be used in conjunction with the `append`, `hash`, `longernewer`, `newer`, or `nodelete` switches.

hash

The `hash` switch allows the transfer of only those files that don't already exist on the destination system or that differ in size or **hash code.** The `hash` switch calculates the CRC32 hash code for the entire file. `hash` cannot be used in conjunction with the `append`, `different`, `longernewer`, `newer`, or `nodelete` switches.

longernewer

The `longernewer` switch allows the transfer of only those files that either don't already exist on the destination system, that are longer, or that have a more recent date and time. `longernewer` cannot be used in conjunction with the `append`, `different`, `hash`, `newer`, or `nodelete` switches.

newer

The `newer` switch allows the transfer of only those files that don't already exist on the destination system or that have a more recent date and time. `newer` cannot be used in conjunction with the `append`, `different`, `hash`, `longernewer`, or `nodelete` switches.

nodelete

The nodelete switch prevents existing files on the destination system from being overwritten. nodelete cannot be used in conjunction with the append, different, hash, longernewer, or newer switches.

recovery

The recovery switch turns on ZMODEM "crash recovery" mode. Crash recovery allows a failed file transfer to resume in midstream or mid-file, without retransferring files or parts of a file that have already been received correctly on the destination system. recovery cannot be used in conjunction with the text switch. If a file specification must be converted to make it legal on the destination system, recovery mode is always automatically disabled to prevent altering files that are not truly identical but that ended up with the same name due to truncation (or other file name alteration).

If a file does not already exist on the destination system, it will always be transferred in its entirety. Files that already exist and are shorter on the destination system will be appended to, beginning with the first byte of the original file beyond the length of the destination file (allowing partly transferred files to resume midstream). Some versions of ZMODEM compare the hash codes of the part of the file that already exists to ensure that the first part of the files' contents are identical. (This capability, known as second-level crash recovery or -rr in some ZMODEM programs, prevents inadvertently tacking a different file onto a like-named, shorter, but completely different existing file). Other ZMODEM programs do not compare hash codes. Autolog will compare hash codes if both the hash and recovery switches are used, if hash comparison is supported by the remote ZMODEM program. Files of the same length (and same hash code, if the hash switch is used) are not transferred. If the destination file already exists but is *longer,* whether the file is transferred or not is decided by the receiving ZMODEM program. Autolog will allow longer files to be completely replaced by shorter ones with zreceive when the recovery switch is used. When using ztransmit, the remote ZMODEM program will determine whether to allow the file transfer or not.

text

The text switch converts the line endings of files to the line endings used by the destination system, because different operating systems use carriage return (control-M), linefeed (control-J), or both to mark the ends of lines in ASCII text files. The way that Autolog will convert line endings depends on your local operating system: Refer to the ***Autolog Installation and Platform Guide*** for details. text cannot be used in conjunction with the recovery switch.

crc16

The `crc16` switch causes CRC16 error-checking rather than the default CRC32 error-checking used for ZMODEM transfers. Use this switch when the remote ZMODEM program does not support the more reliable, default 32-bit error-checking.

encode

The `encode` switch forces Autolog to encode all control characters during a ZMODEM file transfer. Use this switch when your communications channel cannot pass all characters transparently (e.g., when one of the modems involved absorbs the XON and XOFF characters or uses them for flow control) or when the remote system cannot accept certain control characters.

existing

The `existing` switch allows the transfer of only those files that already exist on the destination system. Files that don't already exist are not transferred.

nostamp

The `nostamp` switch suppresses the transfer of the date and time information of files. When downloading files with `zreceive` using the `nostamp` switch, the date and time information of the original file will be discarded, and instead Autolog will mark the file with the current system date and time. When uploading files with `ztransmit` using the `nostamp` switch, Autolog will not send the existing local date and time information; the receiving system and ZMODEM program will determine how the destination file is date- and time-stamped.

query

The `query` switch causes Autolog to prompt you before transferring each file. Respond **`Y`** to transfer the file or **`N`** to skip that file. (Be careful not to delay too long in your response, or the file transfer may time out and abort.)

window

The `window` switch sets the size of the "streaming window," or buffer, of a ZMODEM file transfer. The format of the `window` switch is

`window:`*number*

where *number* is the largest number of bytes that can be transferred before waiting for an acknowledgment. Normally, for maximum speed, the ZMODEM window may be the size of the entire file (meaning that ZMODEM will not look for error-correction information or acknowledgment of the receipt of data until the file is completely

transferred). When the remote system is very busy, with an unreliable modem connection, or with some bursty types of line noise, this can make file transfers very inefficient. Under conditions where data frequently needs to be retransferred, you should set the window size to be smaller. A smaller window means less efficiency on good connections, but better efficiency on bad connections.

Switches for `ztransmit`

In addition to the switches just described, you may use the following switches for selecting files to upload using the `ztransmit` command: the date and time switches (`bdate`, `cdate`, `udate`, `before`, and `after`) and on AMOS computers `contig` and `seq` (note that, although you can select contiguous files to transfer, they will be received on the destination system as sequential files). These switches are identical to the same-named switches for SMT protocol and are explained in detail in the section "Switches for SMT Protocol."

ZMODEM File Transfer Options

There are two options that can be used to control how the ZMODEM file transfer commands `ztransmit` and `zreceive` behave. The third option, `rename`, works only with the `zreceive` command.

`timeout` Option

The `timeout` option adjusts the amount of time Autolog will wait before retransmitting a packet or acknowledgment if no response has been received from the remote system. The default value should work fine under most conditions. To change the timeout value, enter

`timeout` *number*

where *number* is the number of seconds you want Autolog to wait before retransmitting a packet or acknowledgment. See page 54 above for more details about the `timeout` option.

`alarm` Option

The `alarm` option causes a "beep" or "bell" to sound after a ZMODEM file transfer command is completed. If the ZMODEM command used the `q` (query) switch, the alarm is sounded each time you are asked if you want to send a file. Turn the `alarm` option on or off with the command

`alarm` *true or false*

`rename` Option

The `rename` option prevents like-named files from being overwritten on the local computer when using `zreceive`. Instead, a file that has the same name as an existing local file will be renamed, using a convention

that varies depending on your operating system. This option is particularly useful when receiving a batch of similarly named files that may end up having the same name due to truncation or other name conversion.

Turn on the `rename` option with the command

`rename` *true or false*

While the `rename` option is true, when you `zreceive` a file that has the same name as an existing local file, the new file will be renamed. The name it is given depends on your operating system.

UNIX and AIX: The first 26 files will be called `au`*xnnnn,* where *x* is a letter from a to z and *nnnnn* is a five-digit number that corresponds to Autolog's job process number. The next 26 files will be called `bu`*xnnnnn*, and so on. For example, the first 27 files will be renamed `aua00950`, `aub00950`, . . . , `auz00950`, `bua00950`, ...

DOS/Windows: The first 27 files will be called `au`*xnnnnn,* where *x* is the number 0 or a letter from a to z and *nnnnn* is an arbitrary five-digit number that will be the same for the entire Autolog session. The next 27 files will be called `bu`*xnnnnn,* and so on. For example, the first 28 files will be renamed `au010718`, `aua10718`, `aub10718`, . . . , `auz10718`, `bu010718`, `bua10718`, ...

AMOS: The `rename` option is not available on AMOS.

Troubleshooting ZMODEM File Transfers

Check this list if you experience problems using ZMODEM file transfers.

- Be sure the remote system is using ZMODEM protocol, and not some other **file transfer protocol.**
- If you are relying on the remote system to start its ZMODEM program with the "rz" command, check that this command is supported and that the remote system is at a system or shell prompt before attempting a file transfer. You can type the command **`rz`** manually while in talk mode to confirm that this starts up the remote ZMODEM program. (To exit rz, press **^X** 12 times.)
- If the remote system does not honor the "rz" command, be sure to start up the remote ZMODEM program and instruct it to begin transferring files in the appropriate direction *before* pressing the **change key** to enter command mode and entering the `ztransmit` or `zreceive` command.
- If your communications channel is not transparent, that is, it does not transparently pass all 8-bit characters, use the `encode` switch.
- On busy systems, packet-switching networks, or timeshare services, you may need a longer timeout value.

- Be sure the remote system supports a particular setting before using a file transfer switch. When in doubt, avoid use of the switches that you're not sure are supported. Some systems may behave unpredictably if you select an option that is not supported.
- With noisy phone lines or unreliable connections or whenever you experience a great number of retry errors, try selecting a smaller window size with the `window` switch.

If you have a problem that you can't resolve on your own, Soft Machines' technical support can gather useful diagnostic information from the `debug protocol` command. Turn this option on with the command:

```
debug protocol true
```

and try your file transfer again. The `debug protocol` option causes diagnostic information about the file transfer to be displayed on your screen as the file transfer progresses.

3.3 YMODEM File Transfers

YMODEM is an error-correcting file transfer protocol that, although generally less sophisticated than and not as fast as ZMODEM, is also very widespread. A "turbo" variation, YMODEM-g, can be faster than ZMODEM but requires a perfectly clean connection. The `ytransmit` and `yreceive` commands transfer files using YMODEM protocol, and `ygtransmit` and `ygreceive` use YMODEM-g protocol.

You will usually need to prepare the remote system for a YMODEM file transfer while in talk mode, then press the **change key** to enter command mode before issuing a YMODEM file transfer command.

Uploading Files with `ytransmit`

The `ytransmit` command is used to send a file or batch of files to the remote system using YMODEM protocol. Usually, you will need to prepare the remote system to begin receiving a file while in talk mode before using the `ytransmit` command. The steps in performing a `ytransmit` upload are as follows:

1. While in talk mode, prepare the remote system to receive a file using YMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that files will be sent (the remote system will be *receiving* the file), or tell it to use YMODEM rather than another file transfer protocol. The actions you need to begin the transfer and the way to specify files depend on the remote system's YMODEM software. The remote system will then begin signaling that it is ready to receive a file. If Autolog does not begin transmitting the file within a timeout period

determined by the remote YMODEM program (usually about 30 seconds), the file transfer will abort.

2. Press the **change key** to enter command mode. Then enter the command

   ytransmit *file(s)*

   where *file(s)* are the file specification(s) of the file(s) you want to send. Other formats and options for the `ytransmit` command are available and are discussed below.
3. Autolog will now wait up to 60 seconds for the "go-ahead" signal from the remote system. If the delay is too long between starting the file transfer on the remote side and starting it on the local side, the file transfer will fail. Once the "go-ahead" signal is received, the file transfer begins.

As the file transfer progresses, a bar graph similar to the one in Figure 3.6 (or an abbreviated, text-only report if the `terse` option—discussed in Chapter 2—is turned on) will be displayed.

You can specify a single file or use wildcarding symbols (according to the wildcarding conventions supported by your local system) to specify a batch of files to send, as in these examples:

```
ytransmit myfile.lit
  ytransmit ??file
ytransmit myfile[1-20]
```

You may rename files as they are transferred:

ytransmit *new* **=** *old*

☞ The *new* (remote) file specification must be a valid file specification on your local system as well. Otherwise, you must use the "==" syntax described later.

When transferring files between unlike operating systems, Autolog will automatically convert file names that are not legal on the remote system to names that are acceptable by the receiving system. If you want to disable Autolog's automatic file name conversion or to use a new (remote) file specification that is not legal on your local system, use two equal signs:

ytransmit *new* **==** *old*

where *old* is the local file specification and *new* is the file specification for the remote system. Using the "**==**" syntax, you can transfer only one file at a time, and you cannot use any wildcard symbols.

File Transfer Switches for `ytransmit`

You can use the following file transfer switches in a `ytransmit` command:

`q` (query)
date and time switches (`cdate`, `bdate`, `udate`, `before`, and `after`)
`seq` and `contig` (on AMOS computers)

These switches are discussed in greater detail in the section "Switches for SMT Protocol" above.

Downloading Files with `yreceive`

The `yreceive` command is used to receive a file or batch of files from the remote system using YMODEM protocol. You must prepare the remote system while in talk mode to begin sending a file or files using YMODEM before using the `yreceive` command.

The steps in performing a `yreceive` download are as follows:

1. While in talk mode, prepare the remote system to send a file or files using YMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that files will be sent (the remote system will be *sending* the file), or tell it to use YMODEM rather than another file transfer protocol. The actions you need to begin the transfer and the way to specify files depend on the remote system's YMODEM software.
2. Press the **change key** to enter command mode. Then enter the command

   yreceive

   Notice that no file specification is needed in the `yreceive` command: The file specifications are determined by the sending (remote, in this case) system. You may, however, rename files as they are received as discussed below.

As the file transfer progresses, a bar graph similar to the one in Figure 3.6 (or an abbreviated, text-only report if the `terse` option—discussed in Chapter 2—is turned on) will be displayed.

If you want to rename the file(s) as they are received, use this format of the `yreceive` command:

yreceive *new* **= ***

where *new* is the new file specification(s) (e.g., a single file specification or, for a batch of files, a wildcard specification such as ***.new**). Notice that the file specification to the right of the equal sign should always be ***** or ***.***, because the existing, remote files are specified on the remote system when starting the file transfer.

YMODEM File Transfer Options

There are four options can be used to control how YMODEM file transfers behave.

`alarm` Option

The `alarm` option causes a "bell" or "beep" to sound after a YMODEM file transfer command is completed. When the `q` (query) switch is used with the `ytransmit` command, the alarm is sounded each time you are asked whether you want to send a file. Enter the command

`alarm` *true or false*

to turn the `alarm` option on or off.

`image` Option

The `image` option can be used to strip incoming linefeed characters from files received using `yreceive`. This is useful when receiving a file from a remote system that uses both carriage return and linefeed to mark the ends of lines in text file to a local system that uses only carriage returns. Enter the command

`image` *true or false*

to turn the `image` option on or off. When the received copy of a file is doublespaced, use the `image` option to strip linefeeds and prevent doublespacing.

`timeout` Option

The `timeout` option adjusts the amount of time Autolog will wait before retransmitting a packet or acknowledgment if no response has been received from the remote system. Under most conditions, a timeout value of 30 seconds should work fine. To change the timeout value, enter

`timeout` *number*

where *number* is the number of seconds you want Autolog to wait before retransmitting a packet or acknowledgment. See page 54 for more details about the `timeout` option.

`x1k` Option

The `x1k` option instructs `ytransmit` and `yreceive` file transfers to use a larger, 1-kilobyte packet size instead of the default 128-byte packet size. Turn the larger, 1-kilobyte packet size on or off with the command

`x1k` *true or false*

When the 1-kilobyte packet size is in use, the word *`x1k`* appears in the options box of the Autolog command screen. The default state of the `x1k` option can be changed with the Autfix program (see the ***Autolog Installation and Platform Guide*** for details).

YMODEM-g Protocol: `ygtransmit` and `ygreceive`

The YMODEM-g file transfer commands `ygtransmit` and `ygreceive` behave in almost all respects just like their "plain" YMODEM counterparts. However, YMODEM-g protocol does not wait for acknowledgments of correctly received packets. This makes YMODEM-g extremely fast but very finicky. Another difference is that YMODEM-g protocol does not allow for retransferring packets that are not received correctly. This means that `ygtransmit` and `ygreceive` file transfers simply fail, without retries, on any transmission error. This makes YMODEM-g protocol suitable only on perfectly clean connections between error-correcting modems.

The format and syntax of `ygtransmit` and `ygreceive` are exactly like that of "regular" `ytransmit` and `yreceive`. You may follow the directions given in the preceding sections for `ytransmit` and `yreceive`. Just substitute `ygtransmit` for `ytransmit` or `ygreceive` for `yreceive` whenever you want to use YMODEM-g turbo protocol instead of the regular YMODEM protocol.

Troubleshooting YMODEM File Transfers

Check this list if you are experiencing problems using YMODEM or YMODEM-g file transfers.

- Be sure the remote system is using the correct **file transfer protocol,** YMODEM or YMODEM-g, depending on the file transfer command you enter in Autolog.
- Be sure to start up the remote YMODEM program and instruct it to begin transferring files in the appropriate direction *before* pressing the **change key** to enter command mode and entering the `ytransmit`, `yreceive`, `ygtransmit`, or `ygreceive` command.
- Your communications channel must be transparent, that is, it must transparently pass all 8-bit characters, for YMODEM transfers to work. If either modem absorbs XON/XOFF characters or other characters or uses them for flow control, YMODEM transfers cannot be used. Try using ZMODEM protocol (possibly with the `encode` switch).
- You may need a longer timeout value. A timeout value of 30 seconds should work with most systems, but you may need to set the timeout even longer.
- Try turning the `x1k` option on or off. Autolog will usually be able to detect and correct a mismatch in packet size, but under circumstances where it cannot, forcing the correct packet size will correct the problem.

If you have a problem that you can't resolve on your own, Soft Machines' technical support can gather useful diagnostic information from the `debug protocol` command. Turn this option on with the command:

```
debug protocol true
```

and try your file transfer again. The `debug protocol` option causes diagnostic information about the file transfer to be displayed on your screen as the file transfer progresses.

3.4 XMODEM File Transfers

XMODEM is an error-correcting file transfer protocol that, although generally less sophisticated than and not as fast as YMODEM or ZMODEM, is also very widespread. The `xtransmit` and `xreceive` commands transfer files using XMODEM protocol, and `xgtransmit` and `xgreceive` use XMODEM-g protocol.

You will usually need to prepare the remote system for an XMODEM file transfer while in talk mode, then press the **change key** to enter command mode before issuing an XMODEM file transfer command.

You may transfer only one file at a time using XMODEM protocol. Wildcarding symbols therefore may not be used in XMODEM command file specifications.

☞ XMODEM may alter the size (and therefore also the hash code) of files it transfers, because sometimes null characters or control-Z is appended to the end of the file. While this does not affect the usability or readability of the file, it does change the size and hash code.

Uploading Files with `xtransmit`

The `xtransmit` command is used to send a file to the remote system using XMODEM protocol. Usually, you will need to prepare the remote system while in talk mode to begin receiving a file using XMODEM before using the `xtransmit` command. The steps in performing an `xtransmit` upload are as follows:

1. While in talk mode, prepare the remote system to receive a file using XMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that the file will be sent (the remote system will be *receiving* the file), tell it the name of the file, or tell it to use XMODEM rather than another file transfer protocol. The actions you need to begin the transfer and the way to specify files depend on the remote system's XMODEM software. The remote system will then begin signaling that it is ready to receive a file. If Autolog does not begin transmitting the

file within a timeout period determined by the remote XMODEM program (usually about 30 seconds), the file transfer will abort.

2. Press the **change key** to enter command mode. Then enter the command

   xtransmit *file*

   where *file* is the file specification of the file you want to send.
3. Autolog will now wait up to 60 seconds for the "go-ahead" signal from the remote system. If the delay between starting the file transfer on the remote side and starting it on the local side is too long, the file transfer will fail. Once the "go-ahead" signal is received, the file transfer begins.

As the file transfer progresses, a bar graph similar to the one in Figure 3.6 (or an abbreviated, text-only report if the `terse` option—discussed in Chapter 2—is turned on) will be displayed.

You can specify only a single file in an `xtransmit` command. The file name is not transferred between systems, so the name the remote system gives the received copy of the file must generally be specified to the remote XMODEM program when starting the file transfer.

Downloading Files with `xreceive`

The `xreceive` command is used to receive a file from the remote system using XMODEM protocol. You must prepare the remote system while in talk mode to begin sending a file using XMODEM before using the `xreceive` command.

The steps in performing an `xreceive` download are as follows:

1. While in talk mode, prepare the remote system to send a file using XMODEM protocol. You may need to start the remote system's file transfer software, tell it the direction that the file will be sent (the remote system will be *sending* the file), or tell it to use XMODEM rather than another file transfer protocol. The actions you need to begin the transfer and the way to specify files depend on the remote system's XMODEM software.
2. Press the **change key** to enter command mode. Then enter the command

   xreceive *file*

   where *file* is the file specification you want to give the received file on your local system.

As the file transfer progresses, a display similar to the one in Figure 3.7 will be shown. Note that XMODEM protocol does not transfer information about the file size, so the bar graph (or `% complete` message if the `terse` option is active) that indicates how much of the

transfer is complete is not displayed. Instead, the total number of bytes so far received is displayed.

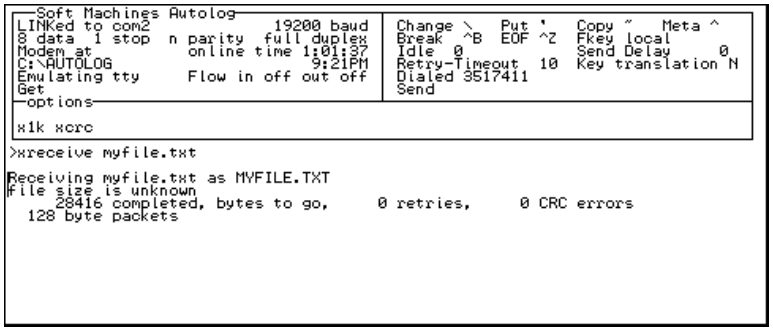


Figure 3.7 `xreceive` file transfer in progress.

XMODEM File Transfer Options

There are five options can be used to control how XMODEM file transfers behave.

`alarm` Option

The `alarm` option causes a "bell" or "beep" to sound after an XMODEM file transfer command is completed. Enter the command

`alarm` *true or false*

to turn the `alarm` option on or off.

`image` Option

The `image` option can be used to strip incoming linefeed characters from files downloaded using `xreceive`. This is useful when downloading a file from a remote system that uses both carriage return and linefeed to mark the ends of lines to a local system that uses only carriage returns. Enter the command

`image` *true or false*

to turn the `image` option on or off. When the received copy of a file is doublespaced, use the `image` option to strip linefeeds and prevent doublespacing.

`timeout` Option

The `timeout` option adjusts the amount of time Autolog will wait before retransmitting a packet or acknowledgment if no response has been received from the remote system. Under most conditions, a timeout value of 30 seconds should work fine. To change the timeout value, enter

`timeout` *number*

where *number* is the number of seconds you want Autolog to wait before retransmitting a packet or acknowledgment. See page 54 above for more details about the `timeout` option.

x1k Option

The `x1k` option instructs `xtransmit` and `xreceive` file transfers to use a larger, 1-kilobyte packet size instead of the default 128-byte packet size. Turn the larger, 1-kilobyte packet size on or off with the command

x1k *true or false*

When the 1-kilobyte packet size is in use, the word *x1k* appears in the options box of the Autolog command screen. The default state of the `x1k` option can be changed with the Autfix program (see the ***Autolog Installation and Platform Guide*** for details).

xcrc Option

The `xcrc` option instructs Autolog to use the more reliable CRC (cyclic redundancy check) error checking instead of the default checksum error checking for XMODEM file transfers. Use the command

xcrc *true or false*

to use CRC (**true**) or checksum error checking (**false**).

Some remote XMODEM programs support only checksum error checking; others may support only CRC error checking; still others support both. For the best results, determine what error correction method is used by the remote XMODEM program and set `xcrc` appropriately. Autolog can frequently detect and try to correct a mismatch of error-checking methods. However, if you experience problems with XMODEM file transfers, try turning the `xcrc` option on or off. The default state of the `xcrc` option can be changed with the Autfix program (see the ***Autolog Installation and Platform Guide***).

XMODEM-g Protocol: `xgtransmit` and `xgreceive`

XMODEM-g is a rarely encountered "turbo" variant of XMODEM protocol. The XMODEM-g file transfer commands `xgtransmit` and `xgreceive` behave in almost all respects just like their "plain" XMODEM counterparts. However, XMODEM-g protocol does not wait for acknowledgments of correctly received packets. This makes XMODEM-g faster than regular XMODEM. Another difference is that XMODEM-g protocol does not allow for retransferring packets that are not received correctly. This means that `xgtransmit` and `xgreceive` file transfers simply fail, without retries, on any transmission error. This makes XMODEM-g protocol suitable only on perfectly clean connections between error-correcting modems.

The format and syntax of `xgtransmit` and `xgreceive` are exactly like that of "regular" `xtransmit` and `xreceive`. You may follow the directions given in the preceding sections for `xtransmit` and `xreceive`. Just substitute `xgtransmit` for `xtransmit` or `xgreceive` for `xreceive` whenever you want to use XMODEM-g turbo protocol instead of the regular XMODEM protocol.

Troubleshooting XMODEM File Transfers

Check this list if you are experiencing problems using XMODEM or XMODEM-g file transfers.

- Be sure the remote system is using the correct **file transfer protocol,** XMODEM or XMODEM-g, depending on the file transfer command you enter in Autolog.
- Be sure to start up the remote XMODEM program and instruct it to begin transferring a file in the appropriate direction *before* pressing the **change key** to enter command mode and entering the `xtransmit`, `xreceive`, `xgtransmit`, or `xgreceive` command.
- Your communications channel must be transparent, that is, it must transparently pass all 8-bit characters, for XMODEM transfers to work. If either modem absorbs XON/XOFF characters or other characters or uses them for flow control, XMODEM transfers cannot be used. Try using ZMODEM protocol (possibly with the `encode` switch).
- You may need a longer timeout value. A timeout value of 30 seconds should work with most systems, but you may need to set the timeout even longer.
- Try turning the `x1k` and `xcrc` options on or off. Autolog will usually be able to detect and correct a mismatch in packet size or error correction method, but under circumstances where it cannot, forcing the correct packet size or error correction method will correct the problem.

If you have a problem that you can't resolve on your own, Soft Machines' technical support can gather useful diagnostic information from the `debug protocol` command. Turn this option on with the command:

```
debug protocol true
```

and try your file transfer again. The `debug protocol` option causes diagnostic information about the file transfer to be displayed on your screen as the file transfer progresses.

3.5 Kermit File Transfers

Kermit file transfers use a communications protocol designed by Columbia University, available for over 200 computer operating systems. Autolog's Kermit file transfers are compatible to other programs that properly follow the Columbia standard.

The Kermit Command

All Kermit functions are handled via a single `kermit` command in `Autolog`. Each different Kermit function is invoked by adding a subcommand to the `kermit` command. For example, to display the current Kermit packet parameters on your screen, you enter the command `kermit` with the subcommand `show`, i.e., `kermit show` **CR**. You may shorten the command `kermit` to just `k`, e.g., `k show` **CR**. Remember to put a space between the `k` and the subcommand, otherwise `Autolog` won't be able to process your Kermit commands!

Throughout the rest of this chapter, we will use the "shorthand" `k {subcommand}` format of the Kermit commands.

Uploading files with `k send`

The `k send` command is used to send a file or batch of files to a remote Kermit. Usually, the remote Kermit must be properly prepared for receiving the files by executing the remote Kermit's RECEIVE command. You can use the `k set autosend` option to do this at the start of each `k send` command.

If the original file name is in a format that is unacceptable on the remote system, it is up to the remote Kermit program to properly alter the file name to a format that is acceptable by the remote system. The remote system operator or your remote system documentation should be able to help you determine what alterations may be made to the file name by the remote Kermit. Some Kermits will report the change of name if this happens, and Autolog will report these remote file name changes to you.

`Autolog` will report the elapsed time for each file transfer, and the effective baud rate of each transfer. If the ALARM option is turned on, your terminal will beep at the conclusion of the file transfer(s).

You can specify a single file or use wildcarding symbols (according to the wildcarding conventions supported by your local system) to specify a batch of files to send, as in these examples:

```
k send myfile.lst
  k send ??file
k send myfile[1-20]
```

You may rename files as they are transferred:

k send *new* **=** *old*

☞ The *new* (remote) file specification must be a valid file specification on your local system as well. Otherwise, you must use the "**==**" syntax to transfer one file. In this case *new* does not have to be a legal filename.

Downloading files with `k receive`

`k receive` is the command to receive a file, or batch of files, from the remote system. The selection of which file or batch of files you will receive is decided by the file specifications given to the remote Kermit's SEND command. The remote Kermit must be already running its SEND command when `Autolog` executes the `k receive` command, or you can use the `k set autoreceive` option to automatically start up the remote Kermit.

To receive files using `Autolog`'s `k receive` command follow these steps:

1. If you have set up an `autoreceive` string to start up the remote Kermit, skip to step 4. Otherwise, while in TALK mode, make sure you have started the remote computer's Kermit program. The command needed to start the remote Kermit program varies from computer to computer. Contact the remote system operator, or your documentation for using the remote system, if you need help with this step.
2. Give the remote Kermit the SEND command, specifying which file(s) you want from the remote system. The format of the remote SEND command and whether or not it can support sending batches of files (wildcarding) or only one file at a time depends on the characteristics of the particular remote Kermit program being used. Contact the remote system operator, or your documentation for using the remote system, if you need help with this step.
3. Press the **change key** to return to `Autolog` command mode.
4. Enter the command `k receive` **CR**.

As each file transfer begins, `Autolog` will display the name of the file that is being received. You can press the **cancel key** to abort the file transfer command.

If you want to rename the file(s) as they are received, use this format of the `k receive` command:

`k receive` *new* **`= *`**

where *new* is the new file specification(s) (e.g., a single file specification or, for a batch of files, a wildcard specification such as **`*.new`**). Notice that the file specification to the right of the equal sign should always be **`*`** or **`*.*`**, because the existing, remote files are specified on the remote system when starting the file transfer and don't need to be specified again in the `k receive` command.

Autolog will report the elapsed time for each file transfer, and the effective baud rate of each transfer. If the ALARM option is turned on, your terminal will beep at the conclusion of the file transfer(s).

Switches for Kermit Protocol

A few switches are available to select files and control how Kermit file transfers behave. File transfer switches can be shortened to the fewest number of unique letters (e.g., query can be abbreviated as q).

The behavior and format of file transfer switches depend on your operating system. See the ***Autolog Installation and Platform Guide*** for additional information on their placement in a k send or k receive command. If the switch is placed immediately after the command, with no intervening spaces, the **/** "switch" character will work on any platform, for example, **k send/t** or **k receive/q**. You can also precede a switch with the usual "switch" character, either **/** or **–**, for your operating system. For DOS, Windows, UNIX, and AIX computers, the switch(es) should come immediately after the command, before any arguments, for example, **k send/newer my*.exe** or **k send –q my***.

With k send you can use the query option to interactively decide which files you wish to send; the date/time switches cdate, udate, bdate, before, and after to select which files to send, and the text switch to adjust line endings of text files.

k receive only supports the append and text switches.

Server Commands

More advanced Kermits get around the tedious problem of having to send commands to both computers for each file transfer by using Kermit **server mode**. Once a remote Kermit is in the server mode, you can make as many file transfer requests in either direction as you like, just by entering the appropriate Autolog Kermit file transfer commands locally. At the conclusion of the file transfer sessions, you send a special "Kermit Server" command to the remote Kermit to end its server session so that the remote computer is able to run other programs. Autolog's Kermit supports the commands needed to transfers files in both directions, a directory command to list files on the remote system, a delete command to erase files on the server, two commands related to the remote working directory, and two commands to end a server session.

You start the server on most remote Kermits by giving the command "SERVER" to the remote Kermit. However, the format of the remote SERVER command, or if it even supports server mode, depends on the particular remote Kermit program being used. Contact the remote system operator, or your documentation for using the remote system, if you need help with this step.

Once the remote Kermit has entered server mode, you will be able to use the Kermit server commands detailed in the following sections.

k send

The `k send` command works in server mode just as it does in regular file transfers, as detailed above, with the following difference: you may make repeated `Autolog` `k send` commands, without having to give the remote side its RECEIVE command each time. You may use standard wildcarding for sending a batch of files, or you may use the == syntax described above to send one file with a remote file name that is legal only for the remote system.

While you are sending files, you may cancel the `send` command by pressing the **cancel key**.

k get

`k get` is the command used to "get" a file or a batch of files from a remote Kermit server. The remote Kermit must support server mode and must have the server engaged. You may specify a "wildcard" batch of files if the remote Kermit supports this, using the wildcarding syntax of the *remote* Kermit.

While you are receiving files, you may cancel the `get` command by pressing the **cancel key**.

k dir

The `k dir {spec}` command tells the remote Kermit server to send a directory listing of files on the remote system. The remote Kermit must be in server mode for this command to work. When you use the `k dir` command, a listing of files available on the remote system will be displayed on your screen.

k delete

`k delete {name}` tells the remote Kermit server to delete the file(s) called {name} from the remote system. The file specification must be given in the format appropriate for the remote system.

k cwd

`k cwd` **{directory}** tells the remote Kermit server to **c**hange your **w**orking **d**irectory on the remote system. The directory specification must be given in the format or syntax appropriate for the remote system.

k bye

The `k bye` command is used to send a special command to the remote server to shut down the server and log out the job that was running the server. (On some single-user systems, it may cause the system to reboot or re-initialize itself.) The remote Kermit must be in server mode for this

command to function. The exact procedure performed by the `k bye` command on the remote system depends on the remote Kermit program being used. In general, `k bye` will cause the remote Kermit to exit server mode, the remote Kermit program to end, and the remote job to be logged out or the system re-initialized.

k finish

The `k finish` command is used to send a special command to the remote server to shut down server operation. This command does not log out the job that was running Kermit, so you can perform other activities on the remote computer. The remote Kermit must be in server mode for this command to work. The exact procedure performed by the `k finish` command on the remote system depends on the remote Kermit program being used. In general, `k finish` will cause the remote Kermit to exit server mode and the remote Kermit program to end, but the remote job will not be logged out, so you can run other programs or perform other tasks on the remote computer.

Note: The `k finish` command is different from `Autolog`'s regular FINISH command. The `k finish` command only sends a message to the remote Kermit to shut down server operation: it does *not* hang up the modem, exit from `Autolog`, etc., like the `Autolog` FINISH command!

Kermit Settings Commands

Kermits generally can communicate without a lot of special preparation because part of the protocol involves a simple negotiation of features that permits widely different Kermits to adapt to the "highest common denominator" of features that both support. However, sometimes it is necessary to change some Kermit settings to allow your Kermit to communicate with other Kermits that operate in special environments, or to work around problems in some Kermit implementations. The `k set` command allows you to make changes to the current Kermit settings so that `Autolog`'s Kermit can communicate with these special Kermits.

k show

The `k show` command displays the current Kermit packet parameters. The packet parameters include the error correction type and the end-of-line character discussed below.

k set

The `k set` command allows you to change the settings used for Kermit. From `Autolog` command mode, you may see a list of all changeable Kermit settings by entering the command `k set` **?** CR .

k set APC {send | receive} "string"

This command sets the autosend string for the send and receive commands. The send APC string should start up the remote Kermit, wait a bit, and send it the receive command. The receive APC string should start up the remote Kermit, and then send the send command. (Autolog automatically appends the user's file request to the end of the k set APC receive string.)

Control characters can be sent by using the meta character (^) in the string. Pauses can be included in the string by using the special control character ^255.

For example, the receive APC command is preset to:

"KERMIT^M%DELSEND "

This example is all that is required to start up many remote Kermits. KERMIT followed by a carriage returns starts up the remote Kermit. The %DEL is the special delay character, which pauses for 1/2 a second. Then, the command SEND and a space end the string. When Autolog executes a RECEIVE command, and the k set autoreceive setting is on, then this string is sent before the file transfer proper starts. In addition, the right hand part of the k receive command is also sent to the remote Kermit, followed by a carriage return. This makes the remote Kermit SEND the files you wish to receive.

k set autoreceive

This command controls the sending of the autoreceive strings to the remote Kermit when a `k receive` command is given. The text sent is determined by using the `k set APC receive "string"` command.

k set autosend

This command controls the sending of the autosend string to the remote Kermit when a `k send` command is given. The text sent is determined by using the `k set APC send "string"` command.

k set blockcheck

This command selects which error correction type `Autolog`'s Kermit will use in Kermit packets. `Autolog` Kermit supports 1- and 2-byte checksum and 3-byte CRC error correction, and defaults to 3-byte CRC unless another error correction type is requested by the remote Kermit. You may select the error correction method by entering the command:

k set blockcheck {num} CR

where {num} is **1**, **2**, or **3** to indicate 1-byte checksum, 2-byte checksum, or 3-byte CRC error correction. Normally, the appropriate blockcheck size is negotiated by the two Kermits when transferring files, so you normally won't need to use this command. Whatever error correction method you select, that selection may be overridden by "negotiation" with the remote Kermit.

k set debug

When enabled, the `debug` option shows technical information concerning the state of the Kermit program, and information about packet traffic. Use this option if Kermit file transfers start to work, but fail for no reason you can discern. This option is initially `off` when `Autolog` is first executed. Enter `k set debug on` **CR** to turn on the debug option, and `k set debug off` **CR** to turn it off again. The `k set debug` command is the equivalent of the debug protocol command. Either command has the same effect.

k set endline

This option allows you to set the default end-of-line character that will be used to end Kermit packets. You will only need to do this when communicating with a few mainframe systems, or when using certain communications networks. The end-of-line character is the character sent at the conclusion of each Kermit packet to facilitate its reception at the remote computer. This option defaults to NO end-of-line character. However, if no end-of-line character is specified, `Autolog`'s Kermit will send a carriage return after the initial packets of a Kermit packet exchange, since this is the end-of-line character most often required by remote Kermits. Once the initial packets have been exchanged, `Autolog`'s Kermit will then use the end-of-line packet that the remote system requests, or no end-of-line character if none is requested by the remote Kermit.

If you need to change the end-of-line character, enter the command:

k set endline {character} **CR**

where {character} is the end-of-line character needed by the remote Kermit. You can specify control characters by using the ^ symbol (or current META character—see Chapter 3). For instance, use **^J** to indicate a control-J character. If you have incorrectly changed the end-of-line character, and need to change it back to no end-of-line character, specify the null (^@) character:

k set endline ^@ **CR**

k set packetsize

The `packetsize` setting determines the size of the packets for Kermit file transfers. Under Autolog, a Kermit packet may be from 10 to 1024 bytes large. `Autolog` will normally default to using 94 byte Kermit packets. You may change this packet size using the command:

k set packetsize {num} **CR**

where {num} is a number from 10 to 1024, indicating the number of bytes per packet you wish to use for Kermit file transfers.

This packet size may be altered by negotiation with the remote Kermit if the remote Kermit does not support the packet size you request. If the remote Kermit does not indicate its packet size, `Autolog` will use 80 byte packets, as recommended by the Kermit protocol specifications.

`k set retries`

This option sets the maximum number of times that a Kermit packet can be re-sent before the file transfer function attempted will be abandoned. The default number of attempts for each packet is twelve. The format of this command is:

`k set retries {num}` **CR**

where {num} is the number of retries to allow for each packet before a file transfer function is aborted.

`k set timeout`

This is the number of seconds that can elapse before Kermit will attempt a retransmission of a flawed or unacknowledged packet. The default `timeout` value is ten seconds. The format of this command is:

`k set timeout {num}` **CR**

where {num} is the number of seconds to wait before re-sending a packet.

`k set packetstart`

This lets you redefine the character Kermit uses to mark the start of a packet. The default character used by most Kermits is control-A. Normally you shouldn't ever need to change this character. However, if you need to change the "start of packet" character to accommodate an unusual remote Kermit or remote system, use the command:

`k set packetstart {character}` **CR**

where {character} is the start-of-packet character needed by the remote Kermit. You can specify control characters by using the ^ symbol (or current META character—see Chapter 3). For instance, use **^B** to indicate a control-B character. `kermitkermit`

The `alarm` Option

The ALARM option can be used with KERMIT file transfers to cause a beep to sound after each file transfer command is completed.

The `rename` Option

The RENAME option can be used when receiving files with KERMIT to automatically rename incoming files if they already exist. See ZMODEM's RENAME option section for details on how the new filenames are generated.

Troubleshooting Kermit File Transfers

Check this list if you experience problems using Kermit file transfers.

- Be sure the remote system is using Kermit protocol, and not some other **file transfer protocol.**
- If the file transfer doesn't seem to start up correctly, or quickly dies, try using the command `k set blockcheck 1` before your first file transfer command. (A few Kermits don't compute checkbytes exactly the same, but all Kermits should handle one byte checksums properly!) This tactic also works with remote Kermits that do not properly negotiate all Kermit negotiation packets.
- If your communications channel is not transparent, that is, it does not transparently pass all 8-bit characters, try using the `strip` option, or make sure that the word length is set to seven. This will force Autolog's Kermit to work in a seven bit mode.
- If you are sending binary files, make sure the remote Kermit is set up to handle binary files. Most Kermits default to TEXT rather than BINARY mode. Try the remote Kermit command SET FILE MODE BINARY prior to your file transfers.

3.6 "No Protocol" File Transfers: `send`, `get`, and `append`

In general, you will usually want to use one of the error-correcting file transfer commands discussed previously in this chapter to transfer files with the remote computer. However, Autolog does support some simple, non-error-correcting methods of sending and receiving text files that do not require the remote computer to support any particular file transfer protocol. One useful application of these "no protocol" commands is to create and store a file of the data received during your communications session using the `get` command.

The `send`, `get`, and `append` commands are very basic ways of transferring text between your computer and the remote system. The `send` command simply sends a text file, one character at a time, to the remote system without any error detection or correction. The `get` and `append` commands similarly capture incoming characters received from the remote system and store them in a text file.

Sending a File with `send`

The `send` command sends a text file, one character at a time, to the remote system. It uses no file transfer protocol, so the remote system does not need any file transfer software. However, the remote system must be

ready to capture the incoming characters into a file using a text editor or similar program.

To send a file, follow these steps:

1. Enter the command

   `send` *file*

   where *file* is the specification of the file you want to send to the remote system. The file transfer does *not* actually begin yet.
2. Next, enter talk mode by pressing the **change key**. On the remote system, start up the text editor or file-capturing program that will store the incoming characters into a file.
3. Then press the **put key** (initially the grave or "backwards" accent, **`**, but may be reassigned using the methods discussed in Section 2.6 of Chapter 2). The **put key** toggles on or off the sending of the file. When you first press the **put key**, Autolog will begin sending the file. Another press of the **put key** will suspend sending the file, and so on. Each time you press the **put key** the message *`(SEND on)`* or *`(SEND off)`* will appear on your screen to remind you of the state of the `send` function.
4. Once the last character of the file is sent, the message *`file transferred`* will be displayed.

If you type any characters while the `send` function is in progress, they will be sent to the remote system, intermingled with the characters of the file being sent, so be careful not to type anything you don't want to appear in the file on the remote system.

☞ If you press the **change key** and return to command mode, the `send` function will be suspended until you reenter talk mode and press the **put key**. However, `send` will be terminated if, while in command mode, you enter a `system` (or `!`) command, `direct`, `pwd`, `cd`, `finish`, `quit`, or another file transfer command (except `get`).

Troubleshooting and Options for `send`

Problems that you may encounter using the `send` function usually stem either from the way the text editor or file-capturing program on the remote system works or from the speed at which characters are sent. Each of these problems is discussed below along with options that can help alleviate the problem.

The `image` option. The `image` option discards linefeed characters from outgoing files. Activate the `image` option with the command

`image` *true or false*

Use this if your local computer uses both a carriage return and a linefeed at the end of lines, or if your local system is UNIX or AIX (which use

only linefeeds), when sending to a computer that requires only carriage returns or when sending to a remote file-capturing program that expects typing from a human user as the usual means of input. When files sent to the remote system are doublespaced or end up as one long line, this indicates that the `image` option should be used.

Screen-oriented text editor problems. A screen-oriented text editor is one that uses extensive screen formatting to make displays look attractive and readable to the user. One problem using a screen-oriented text editor to capture sent files is that it takes a comparatively long time to update the screen when new characters are entered. Autolog may send characters too fast, causing the text editor to lose or garble characters. See the next sections for possible solutions. Another solution is to use a less sophisticated program to capture text on the remote system if one is available.

XON/XOFF protocol. The `send` command will honor XON/XOFF protocol, which is a method used by some computers and programs to signal when they are ready to accept more data and when they are too busy to accept data. Autolog uses the standard XON/XOFF characters: control-S for XOFF and control-Q for XON.

The `delay` option. The `delay` option causes Autolog to pause after sending *each line* of a file. Set the length of the delay with the command

`delay` *number*

where *number* is the number of milliseconds you want Autolog to pause after each line.

Use the `delay` option if characters are lost or garbled when you `send` a file, which may be an indication that characters are being sent too fast for the remote file-capturing program.

The `stall` option. The `stall` option causes Autolog to pause after sending *each character* of a file. The length of the pause is set first using the `delay` command discussed above. Then activate `stall` with the command

`stall` *true or false*

Use `stall` if the `delay` option does not adequately slow down the sending of characters.

Capturing a File with `get` or `append`

The `get` and `append` commands cause Autolog to capture all incoming characters received from the remote system into a local file. This is handy for creating a record of the data received during a communications session. The `get` and `append` commands can also be used to capture screen displays, files, or other information that can be displayed by the

remote system, even if it doesn't exist as a downloadable file on the remote system.

To `get` a file, follow these steps:

1. Enter the command

   `get` *file*
   or
   `append` *file*

   where *file* is the specification of the file you want to create. The only difference between `get` and `append` is that if the specified file already exists, `get` will replace the old file with a new one of the same name (unless the `noerase` option discussed below is active), whereas `append` will append to the end of the existing file. Incoming characters will be written to this file as they are received while you are in talk mode. Because, unless otherwise noted, `get` and `append` behave identically, we will refer to the file opened by a `get` or `append` command as a "`get` file," regardless of which command you actually used to open the file.
2. Next, enter talk mode by pressing the **change key**. If you want to capture a specific display of information, enter the command on the remote system to cause that information to be displayed. If you simply want to create a log file of your session, you can conduct your communications session as usual.
3. The **copy key** (initially the tilde, **~**, but may be reassigned using the methods discussed in Section 2.6 of Chapter 2) can be used to toggle on or off the capturing of characters to the `get` file if you wish. The first press of the **copy key** will suspend writing to the `get` file. Another press of the **copy key** will resume writing to the `get` file, and so on. Each time you press the **copy key** the message *`(GET on)`* or *`(GET off)`* will appear on your screen to remind you of the state of the `get` function.
4. The `get` file will automatically be closed when you `finish` Autolog. If you'd like to close the file sooner, enter the command

   `close`

☞ If you press the **change key** and return to command mode, the `get` function will be suspended until you reenter talk mode. However, `get` will be terminated if, while in command mode, you enter a `system` (or `!`) command, `direct`, `pwd`, `cd`, `finish`, `quit`, or another file transfer command (except `send`).

The `noerase` Option

The `noerase` option prevents accidentally overwriting a file when using `get`. Use the command

noerase *true or false*

to turn the noerase option on or off. While noerase is active, if you specify a file that already exists on your computer in a get command, you will see the message *file already exists*.

The strip Option

The strip option strips the eighth **bit** (also known as the most significant bit or **parity** bit) from incoming characters written to the get file. Use the command

strip *true or false*

to turn the strip option on or off.

If the remote system uses parity, you may need to use the strip option to remove the parity bit from the ASCII characters in a text file.

The guard Option

The guard option prevents most nonprintable characters, such as control characters, from being displayed on the screen or written to the get file. Enter

guard *true or false*

to turn the guard option on or off. When guard is active, only the following control characters are permitted to be written to the get file:

backspace (control-H)	formfeed (control-L)
bell (control-G)	linefeed (control-J)
carriage return (control-M)	tab (control-I)

The guard option is useful on noisy communications channels or with remote system that send undesirable control or screen formatting characters.

☞ The guard option will prevent the displays of screen-oriented software on the remote system from being displayed correctly on your screen, because it will filter out the characters used to format the screen.

The note Command

The note command can be used to insert your own text into a get file. Enter

note get "*text***"**

where *text* is the text you want to insert, enclosed in a pair of quotation marks. If you would like to insert text on more than one line, use the characters **^M** (for carriage return), **^J** (for linefeed), or both to indicate a new line, depending on the end-of-line character(s) used by your operating system.

☞ The `note` command will not automatically append an end-of-line character at the end of your text. Use **`^M`** or **`^J`** at the end (or beginning, or both) of the specified text if you would like the inserted note to appear on a line separate from the captured text from the remote system.

☞ The `note` command is also used to insert notes into the Autolog phone log file and session log file. See Section 1.6 in Chapter 1 for details on this use of the `note` command.

The `emset getmode` Setting

When using a terminal emulation, the `emset getmode` command can be used to control the type of data written to the `get` file. See Section 2.5 in Chapter 2 for more details about the `emset getmode` command.

3.7 File Transfers with EOF Protocol: `post` and `capture`

The post and capture commands can be used to exchange files with computers that have software that honors EOF (end-of-file) protocol. This is not an error-correcting protocol, but is a simple way for computers to indicate the end of the file. The character usually used for EOF is control-Z. If the remote software uses a nonstandard character for EOF, it can be redefined using the methods discussed in Section 2.6 in Chapter 2.

The `post` and `capture` commands require less user interaction than `get`, `append`, and `send`, and so are a little easier to use in script, or `go`, files (see Chapter 4 for more information about `go` files).

Uploading Files with `post`

The `post` command is used to send a file to a remote system that honors EOF protocol. Follow these steps to `post` a file:

1. While in talk mode, start up the remote system's EOF file-capturing software.
2. Press the **change key** to enter command mode, and enter

   `post` *file*

   where *file* is the specification of the file you want to send to the remote system.
3. The file transfer will begin. When the entire file has been sent, the message

   `file transferred`

   will be displayed.

Only text files can be sent using `post`, because other types of files may contain the EOF character.

Downloading Files with `capture`

The `capture` command is used to get a file from the remote system that is sent using EOF protocol. Follow these steps to `capture` a file:

1. While in talk mode, start up the remote system's EOF software and instruct it to send a file.
2. Press the **change key** to enter command mode, and enter

   `capture` *file*

   where *file* is the specification you want to give the file on your system.
3. The file transfer will begin. When it is complete, the message

   `file transferred`

   will be displayed.

Only text files can be downloaded using `capture`, because other types of files may contain the EOF character.

Chapter 4
Automating Autolog with Script Files

❖❖❖

Autolog offers a number of ways to make placing calls easier and more automatic. In this chapter, we first discuss script, or `go`, files that can be used to automate part or all of a communications session. `go` files can be simple, often-repeated sequences of commands that spare you from having to type them manually, or they can be sophisticated "programs" that can intelligently handle unattended communications sessions. In the next chapter, we look at the dialer, which offers a way to create a database of frequently called remote systems that can easily be selected from a menu.

4.1 Script Files and the `go` Command

A script file is simply a file that contains a sequence of Autolog commands. When Autolog executes a script file, it performs the commands contained in the file. You instruct Autolog to execute a script by naming the script file in a `go` command.

Any of the Autolog commands discussed in this ***User's Guide*** can be used in a script file. In addition, there are a number of special commands that provide added capabilities to scripts.

Figure 4.1 shows a sample script file for calling Soft Machines' Update Service. Notice that the first line of the file begins with a semicolon (`;`) character. The semicolon is used to indicate a comment, a nonexecutable line for the benefit of a person reading the file, which is used to explain what is happening in the file. In the last line of the file, a comment appears on the same line as a command. Everything that appears before the semicolon on the line is treated as an executable Autolog command; everything that appears after the semicolon is a comment. For compatibility with UNIX- and AIX-style scripts, a comment line can also begin with the character `#`. The `#` character must be the first character on the line.

```
; call Soft Machines
link com1
modem at
dial 1(217)351-7411 ; dial Soft Machines' number
```

Figure 4.1 Sample script file.

Script files can be created with your text editor or word processor as long as they are text-only files. (Be careful that there are no null

characters [ASCII 0 characters] in your scripts. Null characters will prevent `goto` commands from working.) Script files can be located in whatever account or directory you choose, but the normal location for them is the Autolog installation directory (see the ***Autolog Installation and Platform Guide*** for information about this directory). You may name them whatever you choose as long as the file name includes an extension; we recommend always using the file extension `.ago`, which is the default file extension that Autolog will look for.

To execute a script file, enter the command

`go` *file*

where *file* is the specification of the script file you want to execute. If the file is located in the Autolog installation directory, you don't need to specify the full path, since this is where Autolog will normally look for script files. If the script file is located elsewhere, you must specify the path or account. If the file extension is `.ago`, you don't need to specify the extension, since this is the normal file extension for script files that Autolog will look for. (This is why script file names must always have an extension.) Otherwise you must indicate the complete file name. Suppose, for example, that you create the file in Figure 4.1 in the Autolog installation directory and name it `softm.ago`. To execute this script, you would enter the command **`go softm`**.

Script files can be "nested" up to seven levels; that is, one script can contain a `go` command to call another script, which can contain another `go` command, and so on, until seven script files are active at one time.

Autolog always looks for and executes a special script file, `autolog.ini` (or `autlog.ini` on AMOS computers), when you first start the Autolog program. You can execute a different script file instead of `autolog.ini` when starting up Autolog by naming the script file as an argument to the command you use to start Autolog. For example, if the file in Figure 4.1 is named `softm.ago` and is located in the Autolog directory, you could start up Autolog with the command

`autolog softm` (on UNIX, AIX, or DOS)
or
`autlog softm` (on AMOS)

This would start up Autolog and cause it to immediately execute the script `softm.ago`. The usual `autolog.ini` script is not executed.

Script files execute only while in command mode. If you enter talk mode, script file execution is suspended until command mode is reentered. For this reason, some commands work slightly differently when executed from a script file than when executed as a user-entered command. For example, the `dial` command, which automatically puts you in talk mode when executed as a user-entered command, remains in command mode when executed as a script file command so that the script can continue to execute. Because some commands (such as `get` and

`send`) normally work from talk mode, special techniques are required to make them work from script files. These techniques are discussed later in the section on transferring files.

In the following sections we look at some new commands for use in script files and at some commands we've already discussed when they behave differently as script file commands. Most of the new commands we introduce in this chapter are most useful in script files, although you may find some of them (such as the `lookup` command or the `textsend` command) handy when using Autolog interactively. We also discuss advanced capabilities of script files, such as macros, conditional and nonconditional branching, error detection, and waiting for responses from the remote system. These advanced capabilities make it possible to fully automate a communications session.

4.2 Controlling Script Behavior

Without any other special instructions, a script file normally executes commands one after the other, from top to bottom, in the order they appear in the file. As soon as one command is done, the next command begins to execute. There are many reasons why you may need to adjust this behavior: For instance, you may need to have the script file wait for a bit while the remote system completes a command or process. You may need to have a certain group of commands execute only under some conditions (such as when an error occurs) and another section under other conditions (when everything is proceeding as usual). The commands discussed in this section allow you to control when the script file stops running, what it does when it's done, and what information is displayed on the screen while it executes.

Colon Commands

Most of the colon commands control what information is displayed on the screen while a script file executes. In addition, the `:X` command lets you insert a "stopping point" in a script, and the `:K` and `:`<*prompt*> commands allow you to make interactive scripts.

The first character on the line of colon command must be a `:` (colon) character. These are Autolog's colon commands:

`:S` **Silence.** This command causes the script to execute "silently," that is, nothing is displayed on the screen.

`:R` **Results.** This command causes the results of commands to be displayed on the screen, but not the commands themselves.

`:T` **Trace.** This command causes everything to be displayed on the screen: commands and their results.

`:U` **Untrace.** This command turns off trace mode, and allows the previous `:S` (silence) or `:R` (results) mode to resume.

`:X` **End.** This command makes the script stop executing, regardless of whether there are more commands following in the script file.

`:K` **Keyboard input.** This command causes the script execution to be suspended until a user enters something from the keyboard. This is useful when you need a user to manually type a command, or when you simply want a user to press **Enter** to confirm that a process should begin. This command does not do any checking of the user's input.

`:<`*prompt*`>` **Prompt.** This command causes the text contained in the angled brackets to be displayed on the screen. This is useful if you simply want to provide some feedback to a user about what's happening or if you need to instruct a user what to type before a `:K` (keyboard input) command.

Here's an example of how you might use the colon commands. This example uses the `if` command (introduced later in this chapter) to check whether the `dial` command worked properly.

```
:S (hides the "messy details" of link and modem commands)
;call Soft Machines
:<Preparing port.>
link com1
modem at
:<Port ready: Press Enter to continue>
:K (wait for user to press Enter before continuing)
:R (show response of dial command)
dial 1(217)351-7411 ; dial Soft Machines' number
if err0 = 0 goto Connected
:<Dialing failed. Try again later.>
:X (stop execution here if dial command failed)
Connected:
rest of script file if dial worked normally
```

Figure 4.2 Script file using colon commands.

Screen Formatting with the `xy` Command

The `xy` command allows you to position the cursor and format the screen using the codes presented in Appendix B. Use the command

`xy` *row*`,` *col*

to move the cursor to the row and column indicated by *row* and *col,* respectively.

To format the screen, enter

xy **–***num*,*num*

or

xy **=** *num*

where **–***num*,*num* (or *num*) corresponds to the screen formatting action you want to perform. Appendix B shows the screen formatting codes Autolog recognizes. The number you use in an `xy` screen-formatting command of the second format should be the second number of the angle-bracketed code. For example, the code to erase the entire screen is <–1, 0>. To erase the screen, use the command **xy –1,0** or **xy = 0**.

Not all codes are supported by all types of monitors.

The `abort` Command

The `abort` command controls what happens when an error occurs. It determines what pending script files are canceled if an error takes place. Normally, Autolog lets you use the error registers (discussed later in this chapter) to determine if a command failed to execute as expected and to decide what to do about it. The script file continues to execute, and any pending script files (which can be nested up to seven levels deep) will also continue to execute. Any pending shell or system-level scripts will continue to execute too. But in some situations, it may make more sense to abort pending scripts. Use the `abort` command to determine which scripts are aborted when an error takes place. Use the command

abort *number*

where *number* is **0**, **1**, **2**, or **4**. This is what the different abort settings do:

`abort 0` The default abort state. All scripts continue to execute.

`abort 1` Abort all pending Autolog script files and exit Autolog. Any pending shell or system scripts continue to execute.

`abort 2` Abort all pending Autolog script files, exit Autolog, and abort the pending shell or system script if one is active.

`abort 4` Abort all pending Autolog script files, but do not exit Autolog. A pending shell or system script remains pending.

Under DOS and Windows, `abort 2` does not terminate pending shell or system-level scripts.

The `talk` and `idle` Commands

The `talk` command causes Autolog to enter talk mode (as if a user had pressed the **change key**). This is useful for interactive scripts, when only part of the communication session is automated. For instance, you might want to create a script to set up the communications port, dial a number, enter a name and password, change to a certain directory, then let

the user take over. In this type of script, the `talk` command could be the last command in the script.

If any commands remain to be executed in a script, the script's execution is *suspended* while in talk mode. The `talk` command should be used only when a user will be around to press the **change key** to reenter command mode and allow the script to resume. To have an automated script accomplish tasks that normally are done in talk mode, such as logging on or giving commands to the remote system, use the `say` command discussed in Section 4.5 below.

The `idle` command tells Autolog to monitor how long the communications channel remains idle while in talk mode, and to hang up the modem and terminate the script if the channel remains idle too long. Enter

`idle` *number*

where *number* is the number of seconds the communications channel may remain idle while in talk mode before Autolog will hang up the modem, terminate the script, and finish. For instance, **`idle 300`** would cause Autolog to end the call after 5 minutes (300 seconds) of idle time in talk mode. Use **`idle 0`** to disable the idle option and allow unlimited talk mode time.

The `idle` command is an excellent way to prevent your modems from staying connected unnecessarily long, incurring unnecessary phone charges or tying up modems so others can't use them. If you or other users of your system have a tendency to forget to end modem calls, put an `idle` command in your `autolog.ini` file!

The `sleep` Command

The `sleep` command inserts a pause in a script file for a specified number of seconds. Use the command

`sleep` *number*

where *number* is the number of seconds to pause. The number can include tenths of seconds, for example, **`1.5`** or **`0.1`**.

The `sleep` command can be particularly handy after using the `say` command (discussed in Section 4.5 below) to send commands to the remote system. Autolog script file commands execute quickly, one after another. The remote device or system you're talking to may ignore characters you type (or characters sent by a script file) if it is busy processing a previous command. The `sleep` command is one way to ensure that you don't send commands too quickly for them to be processed by the remote system. Another technique is to use the `peek` command discussed in Section 4.5 below.

When the Script Finishes: `bye` and `chain` Commands

The `bye` and `chain` commands determine what happens when a script file is finished executing.

The `bye` command is used to specify another script file that will execute when Autolog finishes, either because a `finish` command is executed or because an `idle` timeout occurred (see the `idle` command discussed above). The `bye` command simply "sets up" the "bye script" for later execution. The specified script does not actually begin to execute until Autolog finishes. The `bye` command can be useful to make sure that a procedure such as logging off the remote system is completed before ending a session. Use the command:

`bye` *file*

where *file* is the name of the Autolog `go` script you want to be executed before Autolog finishes. This file is just another Autolog script that you've created, with the default file extension `.ago`.

The `chain` command lets you specify a system command or another program to run when the script finishes. The specified command or program begins to execute as soon as the `chain` command is executed.

The scripts in Figures 4.3 and 4.4 illustrate the `bye` and `chain` commands.

```
; call Soft Machines' update system
bye smlogoff.ago
link com1
modem at
dial 217-351-7411
rest of script
finish ; smlogoff.ago executes now
; smlogoff.ago does not need to contain a finish
; command
```

Figure 4.3 The `bye` command.

```
; call Soft Machines' update system
link com1
modem at
dial 217-351-7411
rest of script
chain haltsys ; system command "haltsys" executes
              ; now.
```

Figure 4.4 The `chain` command.

The `lower` Command

The `lower` command "folds" local file name references to lowercase. This command is needed only on UNIX and AIX systems, where file names are case sensitive. This command is intended to make script files written on other platforms compatible with UNIX's and AIX's file-naming conventions. A script written in all uppercase letters on AMOS, for example, can run on a UNIX or AIX system without changing all local file names to lowercase if you add the `lower` command:

`lower` *true or false*

The `lower` command will "fold" local file names that appear in all uppercase letters to all lowercase letters. File names that contain both upper- and lowercase letters will not be changed. Only local file names are affected by the `lower` command, for example, script file names in `go`, `chain`, or , `bye` commands; log file names in `logfile` commands; and so on.

4.3 The `dial` Command in Script Files

The `dial` command normally switches Autolog from command mode to talk mode after a connection is established. This is handy when you have typed the `dial` command by hand, because typically your next task is to log on to the remote computer. However, script file execution is suspended whenever you enter talk mode. It doesn't resume until a user presses the **change key** to reenter command mode. When the `dial` command is executed from within a script file, you often want the script to continue. Sometimes the script may be entirely automated, or you may want the script to complete the log-on procedure for you. Therefore, the `dial` command does not automatically switch from command to talk mode when it is executed from a script file.

4.4 Macros and Registers

Autolog's script files can use two sorts of "variables": macros and registers. A macro is used for storing text; a register is used for storing a number. Autolog provides 10 user-definable registers (plus a few special-purpose registers) and 16 user-definable macros.

Macros

Macros are text "variables." Autolog's 16 macros, $0 through $9 and $a through $f, can contain any sort of printable characters you wish. Once you've defined a macro, you can use a short-hand representation to "plug in" the contents of a macro into an Autolog command. There are three ways to define macros. Once you've defined them, you can reference

them with the dollar sign ($) character plus the number or letter of the specific macro you want to use (0–9 and a–f). For example, you could store a phone number in macro 1. Then you can dial the number with the command **`dial $1`**.

Following are the three ways to assign text to macros.

The `setmacro` Command

The `setmacro` command lets you assign text to a single macro at a time from Autolog's command mode. Enter the command

`setmacro` *id*`:<`*text*`>`

where *id* is the macro identifier—a number from 0 to 9 or a letter from a to f—and *text* is the text you want to assign to this macro, enclosed in a pair of angle brackets. You may omit the angle brackets if the text is a single "word," with no spaces or punctuation characters.

Loading a Macro File with the `macro` Command

You can create a file that contains one or more macro definitions, one per line. The macro definitions contained in the macro file should be in this format:

id`:<`*text*`>`

which is the same format you'd use in a `setmacro` command. Your macro file can define just one or up to all 16 macros. Figure 4.5 shows an example macro file.

```
0:<555-1212>
1:<Selena's computer system>
a:<my log-in name>
b:<my password>
```

Figure 4.5 Sample macro file.

You can create a macro file using a text editor or word processor, but it should be saved as a text-only file. You can call your macro file anything you like, but we recommend that you use the default file extension `.amf` (Autolog macro file).

After creating a macro file, whenever you want to use those macros, you load the file with the macro command:

`macro` *file*

If your macro file is located in the Autolog directory, you don't need to give a path. If the file extension is `.amf`, you can omit the extension too.

Defining Macros When You Start Up Autolog

You can define macros as you start Autolog by including them in your `autolog` (or `autlog`) command. To distinguish macro definitions from

other types of arguments or switches that may appear in the `autolog` command line, they must be preceded by the dollar sign (`$`) character, like this:

```
autolog $ 0:555-1212 1:"log in" f:password
```

Note that there must be a single $ before the list of macro definition(s), there must be a space after the $, and multiple-word definitions must be enclosed in a pair of double quote characters.

Registers

Registers are Autolog's numeric "variables." Autolog provides 10 user-definable registers, `reg(0)` through `reg(9)`; two error registers, `err0` and `err1`; a "stopwatch" register called `timer`; two read-only baud rate registers, `connect'baud` and `serial'baud`; and one read-only platform register, `platform`.

First we'll look at the 10 user-definable registers. Then we'll discuss the special-purpose registers.

User-Definable Registers

Autolog's 10 user-definable registers are called `reg(0)` through `reg(9)`. They provide a way for you to store integer values for counting, controlling the number of times a loop iterates, controlling the number of times a command (such as `dial`) is attempted, and so on. You can perform arithmetic and Boolean operations on the registers. The registers are particularly useful in comparisons made by `if` commands (discussed later in Section 4.5).

The registers are defined using the `set` command. You also use the `set` command to increment or decrement a register. This is the format of the `set` command:

`set reg(`*id***`) =`** *integer or expression*

where *id* is a number from 0 through 9 to indicate which of the 10 registers you want to set, and *integer or expression* is either an integer number or an expression that evaluates as an integer expression. The expression can include addition (`+`), subtraction (`-`), multiplication (`*`), or division (`/`) operations. Because the registers are integers, noninteger results of expressions will be truncated. The expression can also include the Boolean operators AND (`&`), OR (`|`), or exclusive OR (`^`). The Boolean operators treat integers as an array of bits. Valid `set` commands include the following examples:

```
set reg(0) = 5
set reg(3) = 5/3 (result is 1 because of truncation)
set reg(7) = reg(1) & reg (2) | reg(3)
set reg(1) = reg(1) - 1
```

```
set reg(9) = err0
set reg(8) = serial'baud
```

Notice that the last four examples refer to other registers. The fourth example shows how to decrement a register. The fifth and sixth examples show references to other special-purpose registers. `err0` is an error register, and `serial'baud` is a read-only baud rate register.

Error Registers

Autolog has two error registers, `err0` and `err1`, that are used to report errors. When using Autolog interactively, you can tell if an error occurred and what type of problem it was by the error message Autolog reports. The `err0` error register provides a way for script files to be able to detect errors as well. When a command doesn't execute as expected, a number corresponding to the appropriate message is stored in `err0`. Table 4.1 shows the Autolog error codes that can be reported in `err0`.

The `err1` error register is a flag to show whether an error occurred on the local or remote system. `err1` is zero if the error happened locally, and it is nonzero if the error took place on the remote system. For example, when `err0` is equal to 3 and `err1` is not zero, that means a file was not found on the remote system.

Table 4.1 Error Codes Reported in `err0`

*File system errors**

1 File specification error
2 Insufficient free memory
3 File not found
4 File already exists
5 Device not ready
6 Device full
7 Device error
8 Device in use
9 Illegal user code
10 Protection violation
11 Write protected
12 File type mismatch
13 Device does not exist
14 Illegal block number
15 Buffer not initialized
16 File not open
17 File already open
18 Bitmap error
19 Device not mounted
20 Invalid file name
21 BADBLK.SYS has bad hash
22 BADBLK.SYS in bad format
23 BADBLK.SYS not found
24 Insufficient queue blocks
25 MFD is damaged
26 First logical unit not mounted
27 Remote is not responding
28 File in use
29 Record in use
30 Deadly embrace possible
31 File cannot be deleted
32 File cannot be renamed
33 Record not locked
34 Record not locked for output
35 LOKSER queue is full
36 Device is not file structured
37 Illegal record size
38 Block allocate/deallocate error
39 Invalid argument address
40 Invalid argument

Table 4.1 ***continued***

41–155 *Reserved for future use*

Autolog-specific errors

156 Autolog internal flag

157–201 *Reserved for future use*

202 Port is not using IRQ defined in Autfix

203 Divide by zero

204 Terminated by a signal

205 Macro expansion terminated

206 Action already defined

207 Action not defined

208 Port already open

209 Feature not available

210 Out of memory

211 Output in progress timeout

212 Source file is older

213 Source file is shorter and older

214 Table is full

215 Modem driver does not support this feature

216 Files have same hash code

217 Files have same version

218 Unexpected packet type

219 Batch transfer canceled

220 File transfer canceled

221 Miscellaneous file transfer error

222 Program must be in system memory for badly designed I/O ports

223 Interface driver does not support this feature

224 No emulation driver

225 File transfer failed

226 File transfer timed out

227 Missing label in `goto` command

228 End of file

229 Remote system cannot support wildcarding for this command

230 CMDLIN.SYS required for this command (AMOS)

231 Illegal macro expression

232 Macro expansion failed

233 Syntax error

234 Quoted string had bad form

235 No match received

236 No user activity timeout

237 Modem does not respond

238 Call aborted†

239 No answer†

240 Line is busy†

241 No dial tone†

242 Maximum `go` file level exceeded

243 No modem driver

244 Interface driver or COMSER.IDV required but not found

245 Invalid baud rate

246 Not linked to remote port

247 No argument given

248 Invalid argument

249 Command error

250 Not found

251 Remote system does not respond

252 Loss of modem carrier

253 Transfer retry count exceeded

254 Transfer check count error

255 User interrupt

*Some of these errors are specific to the AMOS platform and will not be reported by other operating systems.

†Not all modems are capable of reporting all dialing errors. For modem-independent scripts, check for all these error codes or for just a nonzero `err0`.

Timer Register

The timer register acts as a stopwatch for timing events. The timer measures time in seconds. Start the timer with this command:

set timer = *number*

where *number* is the starting time in seconds (usually zero). You can then check the timer value in an `if` command (discussed in Section 4.5 below) such as this:

```
if timer > 300 goto EndCall
```

The timer continues to count seconds from the time you set its value until you `finish` out of Autolog or assign it a new value with the `set` command.

Baud Rate Registers

Autolog's two baud rate registers store the baud rate of the communications port (`serial'baud` register) and of the modem connection (`connect'baud` register). These are read-only registers: You can't change them. You can use these baud rate registers in `if` commands, as in these examples:

```
if serial'baud < 19200 baud 19200
if connect'baud = 9600 baud 19200
```

Platform Register

The `platform` register indicates the operating system under which Autolog is running. The `platform` register is read-only register whose value is an integer that indicates the operating system, as follows:

1 AMOS
2 SCO-UNIX
3 AIX
4 DOS or Windows 3.x
5 Windows 95 or Windows NT (WIN32 platforms)

4.5 Waiting for Responses and Looping

Normally the "dialog" between your computer and the remote system takes place in talk mode. For instance, when you first connect to the remote system, you enter talk mode, where you may be prompted to enter a password, which you type from your keyboard. In order to automate such an exchange, your script file needs a way to determine what the remote computer sends to your computer and a way to send responses back. Autolog offers several commands to help your scripts wait for prompts, send responses, and determine whether the remote system has responded in an expected way.

The `say` Command

The `say` command lets a script send characters to the remote system, simulating a user typing them interactively from the keyboard while in talk mode. For example, when you first connect with a remote system, you may be prompted to enter a log-in name. Normally you'd type your log-in name and press the "return" or "enter" key. You can use the `say` command to "type" your name and the "enter" key. The `say` command should be in this format:

`say "`*`text`***`"`**

where *text* is the characters you want to send to the remote system, enclosed in a pair of double quotes.

You can include nonprintable characters in the `say` command by preceding them with the **meta** character. For example:

`say "^C"`

sends a control-C character to the remote system. Non-ASCII characters may be included by typing the **meta** character plus the decimal value of the character; for example:

`say "^155"`

sends an ESC character with the high bit set (27 + 128) to the remote system. You may also say the *contents* of a macro; for example:

`say "$1"`

To send a dollar sign, enter two dollar signs: **`say "$$"`**; this is necessary to distinguish a "real" dollar sign from a macro.

The `say` command does *not* automatically append the **Enter** character to the end of a command. If you want a "return" or "enter" character, include the characters ^M (control-M—use the current **meta** character if it's not currently the ^ character) within the double quotes in the place where you'd press **Enter**: for example, **`say "password^M"`**.

The `goto` Command

The `goto` command works in conjunctions with *labels* that you insert in your script file. Normally a script is executed from top to bottom, one command after another in the order they appear in the script. The `goto` command gives you a way to "jump" from one place to another in the script.

A label is simply a name followed by a colon that is inserted on a line by itself at the place you want to jump to. A label can consist of one or more alphanumeric characters followed by a colon. The label must fit on one line of the script file. It can contain upper- and lowercase letters,

numbers, and most punctuation characters (except the colon), but it must not contain any spaces. Labels are case sensitive, so **`Label:`**, **`label:`**, and **`lAbel:`** are all distinct labels. Here are some examples of well-formed labels:

```
dial'again:
Transfer_Files:
do.this.1st:
1:
```

After a label has been inserted in a script, you can add the `goto` command to have the execution of the script jump to the next command after the label. The `goto` command has this format:

`goto` *label*

Be sure that the *label* named in a `goto` command matches an existing label exactly.

The sample script in Figure 4.6 shows how you can use the `goto` command to create a loop. The `if` command (which is discussed in detail a little later) checks to see whether there was an error after dialing. This dialing loop will dial a phone number, then check to see whether the `dial` command executed without a problem (`err0` will be zero if the dial worked; it will be nonzero if there was a problem such as "call aborted," "no answer," etc.). If the `dial` command failed for any reason, the `goto` command will redirect the execution of the script back to the `dial` command (the `try'dialing` label) so dialing can be tried again. This loop is rather simple: It will execute infinitely if for some reason the `dial` command never executes properly (for instance, if you typed the wrong phone number so a person rather than a modem answers!). A little later we'll look at ways to make this loop a little better by limiting the maximum number of times it will execute.

```
try'dialing:
      dial 555-1212
      if err0 = 0 goto dialed'ok
      goto try'dialing
dialed'ok:
      more commands to sign on, etc.
```

Figure 4.6 Simple dialing loop.

The `if` Command

The `if` command can be used for making decisions based on the values of registers, for checking to see whether a particular response has been received from the remote system, or wherever conditional branching can be useful.

The `if` command has this basic format:

`if` *condition command*

The *condition* is either an expression that can be evaluated as true or false (e.g., `reg(1) = 3`) or text within a pair of double quotes. Because these two types of `if` commands are rather different, we'll discuss each separately.

`if` with True or False Condition

This type of `if` command evaluates a statement involving a comparison of integers (e.g., `reg(1) = 3`) as either true or false. If the statement is true, the command given in the `if` is executed. If the statement is false, the command is not executed. Execution of the script then proceeds to the next command (or the next command after the label if the condition is true and the command indicated is a `goto` command).

The condition that the `if` evaluates is a comparison of integer values. This is where Autolog's registers come in handy. The condition can reference any of Autolog's registers and can use these comparisons of equality: = (equal); #, <>, or != (not equal); > (greater than); < (less than); <= (less than or equal to); or >= (greater than or equal to). Here are some examples of valid `if` commands:

```
if connect'baud = 9600 baud 19200
    if err0 # 0 goto had'error
if timer <= 300 goto got'time'left
if reg(0) >= 1 reg(0) = reg(0) - 1
```

Now let's revisit the dialing loop that first appeared in Figure 4.6. In Figure 4.7, we use the `reg(0)` register as a loop counter. Before we

```
set reg(0) = 3    ; loop counter: try 3 times
try'dialing:
   if reg(0) = 0 goto dial'failed
   dial 555-1212
   if err0 = 0 goto dialed'ok
   reg(0) = reg(0) - 1  ; decrement loop counter
   goto try'dialing
dialed'ok:
   more commands to sign on, etc.
   :X
dial'failed:
   note log "dialed unsuccessfully 3 times^M"
   finish
```

Figure 4.7 Improved dialing loop.

begin the dialing loop, we set `reg(0)` to 3. Before we try dialing, we use an `if` command to check that `reg(0)` is not equal to zero. Each time the `dial` command fails (another `if` checks whether `err0` is equal to zero), we decrement `reg(0)` and try again.

if to Check for Response

A second type of `if` command checks whether a particular response or string of characters has been received from the remote system. This type of `if` command uses the format

if "*string***"** *command*

where *string* is the character(s) for which you're checking, enclosed in a pair of double quotes, and *command* is the Autolog command to execute if the string was received. Just as in the `say` command, you can include control characters, non-ASCII characters, or the contents of a macro in the string, as in these examples:

```
if "^M" goto line'end'received
  if "$2" say "My password^M"
if "^155" goto got'high'bit'escape
```

To understand this type of `if` command, it will help to understand about Autolog's *holding buffer,* a storage area where Autolog stores characters received from the remote system.

As characters are received from the remote system (for instance, a log-in prompt when you first connect to the remote system), they are placed in the holding buffer. The characters you send to the remote system are stored only if the remote system echoes them back (full duplex). The holding buffer is limited in size (about 3,000 bytes). If the holding buffer fills up, the oldest characters at the beginning of the buffer are thrown away to make room for new characters coming in. The holding buffer can hold from about one quarter to one whole screenful of information, depending on the type of data and screen formatting sent by the remote system.

When an `if "`*string*`"` *command* command is executed, Autolog searches in the holding buffer for the sequence of characters represented by *string.* If the characters are found within the holding buffer, the contents of the holding buffer *up to and including the matched string are thrown away* from the holding buffer, and the specified command is executed. If the characters are not found in the holding buffer, the holding buffer is not changed and the command is not executed.

☞ Hardware flow control (see the `flow` command in Chapter 1) can sometimes create a special problem for an `if` command in a loop. If the holding buffer fills up, no more characters can be received, and Autolog uses the flow control signal to prevent more characters from being accepted. But if the holding buffer isn't emptied in some

way, no new characters can be accepted, so the match string in the `if` command will never be received, no matter how many times the loop containing the `if` command is repeated. To avoid this problem, add an `until` command (discussed below) right after your `if` command. The `until` command will discard some characters from the holding buffer, allowing new characters to be received again.

Example. To better understand how the `if` command works, imagine this scenario: After placing a call to a remote modem, you must press **Enter** to get the remote system's attention. The remote system then sends the prompt *`Enter name:`* to which you type the log-in name **`guest`**. Without a way to wait for the expected prompt, a script file to log on to this system might look like this:

```
dial 555-1212
say "^M"          ; get remote's attention
say "guest^M"     ; give log-in name
```

This script may work quite well on some systems, but it may fail on others if the log-in name "guest" is typed too soon, before the remote system has sent the *`Enter name:`* prompt. We can improve the script by adding an `if` command to check that the prompt has been received before we give the log-in name:

```
dial 555-1212
say "^M"          ; get remote's attention
look'for'prompt:
if "Enter name:" say "guest^M"
if err0 # 0 goto look'for'prompt
```

In this example, the `look'for'prompt` loop is necessary to ensure that the prompt has been received. Consider the case in which the `if` command is executed at the moment that the holding buffer contains the characters "Enter n". Although the prompt is "on its way," the complete string "Enter name:" hasn't been received yet, so the `if` command will fail. Although this loop is a slight improvement, we may still run into problems. When the `if` command fails, we don't know whether the problem is that we just haven't received the prompt in its entirety yet (in which case another iteration or two of the loop will succeed) or whether the prompt will never be received (if the remote system is "dead" or if it sends an unexpected message). The `until` command discussed a little later provides a better way to wait for a prompt.

Clearing the Holding Buffer

A final type of `if` command can be used to empty out the holding buffer. Enter

`if ""`

to empty out the complete contents of the holding buffer. The holding buffer is also emptied by the `peek` command (discussed later in this chapter).

☞ Once the holding buffer has been emptied, any responses received from the remote system until that point are "erased from Autolog's memory." Future `if` or `until` (discussed next) commands that search for responses that have already been received may fail if you clear them from the holding buffer.

The `until` Command

The `until` command provides a way to wait for a response (unlike the `if` command, which can be used only to see if the response has *already* been received). The format of the `until` command is

`until` "*string*" *number*

where *number* is the number of seconds you want to wait for the characters indicated by *string* (which are enclosed in a pair of double quotes).

Example. To return to the example introduced in the earlier section on the `if` command, suppose we want to dial a remote system, press **Enter**, wait for the prompt "Enter name: " and then enter "guest" as our log-in name. We can use the `until` command to wait a reasonable amount of time for the "Enter name: " prompt, as in this script:

```
dial 555-1212
say "^M"          ; get remote's attention
until "Enter name:" 10
if err0 # 0 goto no'prompt
say "guest^M"
; rest of script if success
more commands
finish
no'prompt:
note log "Expected log-in prompt not rec'd"
finish
```

The `fold` and `strip` Commands

The `fold` and `strip` commands make the `if` and `until` commands a little easier to work with by making it unnecessary to match the case and **parity,** respectively, of the received characters. The `fold` option can be turned on or off with the command

`fold` *true or false*

The `strip` command is discussed in more detail in Section 3.5 of Chapter 3. It is turned on or off with the command

`strip` *true or false*

For `if` or `until` commands to succeed, the `"`*string*`"` argument normally must match the received characters exactly, including matching the case and the parity of the received characters. For example, if the remote system sends the prompt *`Enter name:`*, an `if` or `until` command with the match string `"Enter Name:"` would not work because of the uppercase "N." Similarly, if the remote system sends an uppercase "K" using odd parity, waiting for the string `"K"` in an `if` or `until` command will not work (you'd need to wait for `"^203"`: uppercase K plus 128 for the odd parity bit, or 75 + 128). Rather than worrying about matching the case or calculating the parity for every character, you can turn on the `fold` and `strip` options.

`fold` will treat all characters as if they have been "folded" to uppercase for comparison purposes. It will *not* change the actual characters in the holding buffer or written to a `get` or log file. However, `strip` will "strip" the parity bit from characters, including those written to files, as discussed in Section 3.5 of Chapter 3.

The `peek` Command

The `peek` command lets your script wait, not for a specific prompt, but until the remote system stops sending characters. Also, unlike the `if "`*string*`"` and `until` commands, `peek` displays the received characters on the screen. The `peek` command is useful when you're not sure what characters the remote system will send (for example, after sending a command to the remote system to cause it to display a list of files in a directory). It is also useful for scripts during which you want a user to be able to see "what's happening": what characters are being received from the remote system.

Enter the `peek` command in this format:

`peek` *number*

where *number* is the number of seconds of "silence" to wait for. `peek` will display the characters received from the remote system until no more characters are received for specified number of seconds. The `peek` command displays all the characters already in the holding buffer or added to the holding buffer as it executes. `peek` empties the holding buffer.

4.6 Transferring Files in a Script

Most file transfer commands work in script files just as they do during interactive Autolog use. You may find `lookup` handy for checking to see whether a file exists before attempting a file transfer.

You can include any file transfer command in a script file. (File transfer commands are discussed in Chapter 3.) However, normally `get` and `send` work in talk mode and require a user's input (pressing the **put key** or **copy key**) to work. The `press` command allows a script to "press" Autolog's **copy key** and therefore allows `get` to be operated from command mode and in script files. The `textsend` command is a `send` file option that is useful for record-oriented files and handy in script files.

The `lookup` Command

The `lookup` command lets you check to see whether a *local* file exists. Enter

`lookup` *file*

to check for the existence of the specified *file*. `err0` will be equal to 3 (*`file not found`*) if the file wasn't found, and it will be 0 if the file exists.

☞ To check whether a *remote* file exists, you can use the `say` command to send a "dir" or similar command to the remote system, then use an `until` command to wait for the file name to show up in the remote system's response. Of course, if you try to get a file from the remote system that doesn't exist, your file transfer will fail with a *`file not found`* error (`err0` = 3). If you just want to make sure you don't overwrite an existing file, use the `noerase` option (described in Chapter 3).

Using `get` in Scripts: The `press` Command

The `get` command (discussed in Chapter 3) normally operates from talk mode: You enter talk mode and press the **copy key** to begin the capture. By using the `press` command, you can make a `get` file transfer work from command mode (and so from a script file).

To complete a `get` file transfer in a script file, first include the `get` command. Then be sure to `say` the appropriate commands to prepare the remote system for sending or displaying a text file. Finally, use the command

`press copy on`

to begin the file transfer. You can simulate subsequent presses of the **copy key** with the command:

press copy *on, off, or toggle*

where *on* causes `get` to begin or resume, *off* causes `get` to suspend, and *toggle* causes `get` to switch states (from storing incoming characters to not storing them, or vice versa).

You can also use the `press` command to simulate a user's keypress of the **put key**. This does *not* make `send` operate from command mode, however. The `press put` command can be used merely to have `send` begin immediately when a user enters talk mode. The next section on the `textsend` function explains how to use `send` in a script file.

The `textsend` Command

The `textsend` command can be used in conjunction with `send` to send record-oriented files. The `textsend` function modifies the way the `send` command works (discussed in Chapter 3) so that record-oriented files can be sent from scripts.

If you need to send a simple text file with "no protocol," use the `post` command rather than `send` in your script. The `textsend` option can be used with `send` to send files that are organized in "records." Records in this situation are simply chunks of text in a text file that end with a unique terminating character or string of characters. The terminating character may be simply a line ending, or it may be a special flag such as "ENDREC" or a unique control character.

To use the `textsend` function, first include a `send` command in your script. Just as when using Autolog interactively, the `send` transfer does not begin right away: The `send` command simply establishes which file will be sent. You use the `textsend` command to actually cause a record from the file to be sent:

textsend "*record terminator***"**

This command causes the contents of the `send` file to be sent to the remote system up to (but not including) the first occurrence of the text *record terminator*. The *record terminator* text is not sent: You need to use the command **say "***record terminator***"** after the `textsend` command if you need to send the record terminator to the remote system rather than discarding it.

The `textsend` command sends one record at a time. You'll need to build a loop to send all the records in the file. If you know in advance how many records are in the file, you can use a register to count the iterations of the loop to make sure all the records are sent. Otherwise, you can check for the error code 228 (end of file) in `err0` after each `textsend` command.

In Figure 4.8 is an example script that uses the `textsend` option.

```
send records.txt
say commands to start the remote system's file capturing software
send_loop:
      textsend "ENDREC"
      if err0 = 228 goto done_sending
      say "ENDREC"
      goto send_loop
done_sending:
      rest of script
```

Figure 4.8 Using the `textsend` function.

4.7 Debugging Scripts

As a general rule, if you want to check how your script file is working, it's handy to use terse mode (use the command **`terse true`**) and use the **`:T`** (trace) command at the top of your script file to have all commands and responses displayed as they're executed. The Autolog log file, described in Section 1.6 of Chapter 1, can also be used to help diagnose problems with script files. There are also a number of special commands that can be used to help test and debug scripts.

The `show` Command

The `show` command displays the contents of the macros, registers, and holding buffer of received characters. You can enter **`show reg`** to see only the register contents, **`show macro`** to see just the macro contents, **`show buffer`** to see just the number of characters in and contents of the holding buffer, or **`show`** to see all of these.

This command can be helpful to check that the macros and registers contain what you expect. It is also helpful for checking the error registers, `err0` and `err1`, to see if an error occurred in the previous command. It is also useful for checking the contents of the holding buffer if your `if` or `until` commands aren't working as you expect.

Single-Stepping through a Script

Single-stepping through a script file means executing commands one at a time: When you single-step through a script, Autolog will not proceed to the next command until you give the signal to proceed. This slows down the execution of the script to help you analyze what's happening and to help you see which commands are working as you expect and which ones may need some adjustment. You can single-step through an entire script or through only a portion.

The commands to help you debug a script are `debug script`, `haltpoint`, `sstep`, `resume`, and the **pause key**. You can prepare a script for debugging by adding single-step commands to the file, or you can use the single-step commands "on the fly" to debug a script. Whether or not debugging commands are included in a script file, single-stepping does not occur until you turn on the `debug script` option with the command:

debug script true

If `debug script` is not enabled, scripts work normally. This allows you to embed single-step commands into scripts for diagnostic purposes that won't affect the normal operation of the script until you enter debugging mode.

Preparing a script file for debugging:

1. The `haltpoint` command can be inserted at the beginning of a script or at any point where you want to begin single-stepping. You can insert a `haltpoint` command just before a problematic area of the script.
2. The `resume` command can be inserted at the end of the problematic area to resume normal execution of the script.

Debugging a prepared script:

1. Before starting the script file, enter the command

   debug script true

   to enable script debugging. Then enter

   go *file*

   where *file* is the name of the script file you want to debug.
2. When the `haltpoint` command is encountered, execution of the script is suspended until you enter the command **sstep** or press the **pause key**. When you **sstep** or press the **pause key**, the next single command in the script file is executed. You can continue to **sstep** through the script, or you can enter the command **resume** to resume normal execution of the script.

Debugging a script "on the fly":

1. Before starting the script file, enter the command

   debug script true

   to enable script debugging. Make sure to define a **pause key** (see Section 2.6 in Chapter 2). Then enter

   go *file*

   where *file* is the name of the script file you want to debug.

2. When you want to begin single-stepping through the file, press the **pause key**. Execution of the script file is suspended until you enter the command **sstep** or press the **pause key** again. When you **sstep** or press the **pause key**, the next single command in the script file is executed. You can continue to **sstep** through the script, or you can enter the command **resume** to resume normal execution of the script.

While single-stepping through a script, you can enter your own diagnostic commands. For instance, if an `if` or `until` command fails, you can enter the command **show buffer** to display the contents of the holding buffer before entering **sstep** to proceed to the next script file command. It is very helpful to use the `logfile` command (see Section 1.6 in Chapter 1) to create a log file when testing or debugging scripts. We also recommend that you add a `:T` command to script files you want to test (to display all commands and responses) and use the `terse` command for an easy-to-read display and log file.

Chapter 5
The Dialer Menu System

The dialer menu system allows you to store a "telephone directory" of frequently called remote sites, along with useful information about the settings to use and actions to take when calling each site. The dialer presents a menu of sites, allowing you to easily select and call a remote system. Complete instructions for creating and installing the data file and other resources used by the dialer menu system are in the ***Autolog Installation and Platform Guide***. Once the dialer has been installed, you can use the `dialer` command to present the menu and select a site to call.

5.1 Editing the Dialer Data File

To use the dialer menu system, you must first have a data file that contains the sites you want to appear on the dialer menu. This file, `dialer.dat`, should be in your Autolog installation account. It contains an entry for calling Soft Machines' update system. You'll probably want to add entries for your own frequently called sites. You can use your regular word processor or text editor to add entries to this file, but remember to save the `dialer.dat` file as a text-only file.

Each entry in the `dialer.dat` file appears on a separate line. Each entry should appear in this format:

name → *phone number* → *baud rate* → *settings* → *command(s)*

Each field should be separated from the next by a tab (which we'll indicate by →). For example, the menu entry for Soft Machines appears like this:

```
Soft Machines → 1-217-351-7411 → 0 → 81NF → emulate autgen
```

This is what should appear in each of the fields:

Name. The name is text that will appear in the dialer menu. The name can be up to 25 characters long. One or more tabs should separate it from the next field.

Phone number. The phone number should be entered just as you would type it in a `dial` command. It may contain special dialing characters or dashes or parentheses for readability. It should be separated from the next field by a tab.

Baud rate. This is the baud rate of the serial port. If you want to use the normal serial port baud rate (you don't want to change it), use the number 0 for the baud rate.

Settings. This field sets data, stop, and parity bits and the duplex (full or half) in a condensed form. There should be four characters in this field: The first is the number of data bits (e.g., 7 or 8), the second is the number of stop bits (e.g., 1), the third is a letter indicating the parity (N for none, E for even, or O for odd), and the fourth is a letter indicating the duplex (F for full or H for half). For example, **`81NF`** means 8 data bits, 1 stop bit, No parity, and Full duplex. (See Section 1.3 in Chapter 1 and Section 2.2 in Chapter 2 for more information about these port settings.) This can be the last field in a dialer menu entry, or you may include the optional commands field. If the commands field is included, a tab should follow the settings field.

Commands. This field is optional and may be omitted. In this field you can specify a command that will be executed after dialing this site. For example, the Soft Machines' entry uses the command **`emulate autgen`** to select a terminal emulation. You could enter a **`go`** *file* command to execute a script (scripts are discussed in Chapter 4).

If you don't want to create a `go` script for this site, you can include multiple commands in this field by using the `mlt` (multiple) command. The `mlt` command lets you execute several Autolog commands on a single line. The format of the `mlt` command is

`mlt` *first Autolog command* **`|`** *second Autolog command* **`|`** …

After `mlt`, you may include as many commands as you wish, each separated from the next with the vertical bar (or "change bar," `|`).

After you've added your entries to the `dialer.dat` file, be sure to save the `dialer.dat` file in text-only format. Then you're ready to use the dialer menu system.

5.2 Using the Dialer Menu

To use the dialer menu, start up Autolog and enter the command **`dialer`**. A screen like the one in Figure 5.1 will be displayed. You can use your arrow keys to position the cursor on an entry, or type **`S`** to search for an entry. If you change your mind about using the dialer, type **`Q`** to quit the dialer menu system and return to Autolog's command mode. When the cursor is resting on the site you want to call, press **Enter** to place the

call to that site. Autolog will dial the site using the settings specified, and then will execute the commands (if any) specified in the entry. You will then be able to access the remote site as usual.

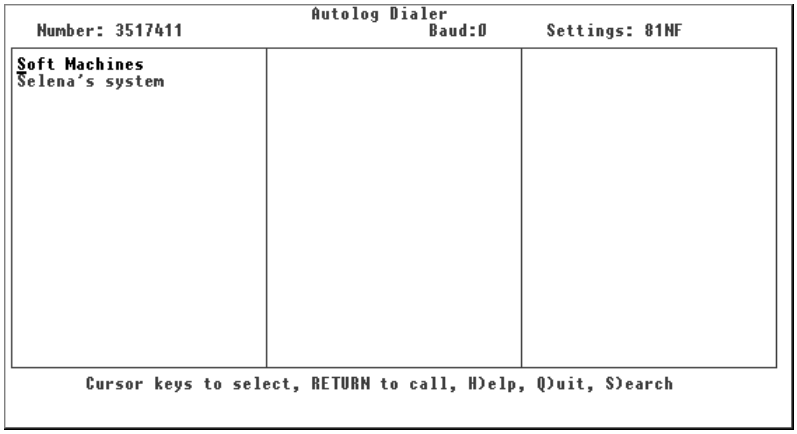


Figure 5.1 The dialer menu.

Appendix A
ASCII Characters

ASCII	*Decimal*	*ASCII*	*Decimal*	*ASCII*	*Decimal*	*ASCII*	*Decimal*
NUL	0	SP	32	@	64	‘	96
SOH	1	!	33	A	65	a	97
STX	2	"	34	B	66	b	98
ETX	3	#	35	C	67	c	99
EOT	4	$	36	D	68	d	100
ENQ	5	%	37	E	69	e	101
ACK	6	&	38	F	70	f	102
BEL	7	’	39	G	71	g	103
BS	8	(	40	H	72	h	104
HT	9	)	41	I	73	i	105
LF	10	*	42	J	74	j	106
VT	11	+	43	K	75	k	107
FF	12	,	44	L	76	l	108
CR	13	-	45	M	77	m	109
SO	14	.	46	N	78	n	110
SI	15	/	47	O	79	o	111
DLE	16	0	48	P	80	p	112
DC1	17	1	49	Q	81	q	113
DC2	18	2	50	R	82	r	114
DC3	19	3	51	S	83	s	115
DC4	20	4	52	T	84	t	116
NAK	21	5	53	U	85	u	117
SYN	22	6	54	V	86	v	118
ETB	23	7	55	W	87	w	119
CAN	24	8	56	X	88	x	120
EM	25	9	57	Y	89	y	121
SUB	26	:	58	Z	90	z	122
ESC	27	;	59	[	91	{	123
FS	28	<	60	\	92	|	124
GS	29	=	61	]	93	}	125
RS	30	>	62	^	94	~	126
US	31	?	63	_	95	DEL	127

Appendix B
Screen Formatting Codes

Code	*Purpose*
<-1,0>	Clear screen and set normal intensity
<-1,1>	Cursor home (move to 1,1)
<-1,2>	Cursor return (move to column 1 without linefeed)
<-1,3>	Cursor up one row
<-1,4>	Cursor down one row
<-1,5>	Cursor left one column
<-1,6>	Cursor right one column
<-1,7>	Lock keyboard
<-1,8>	Unlock keyboard
<-1,9>	Erase to end of line
<-1,10>	Erase to end of screen
<-1,11>	Enter background display mode (reduced intensity)
<-1,12>	Enter foreground display mode (normal intensity)
<-1,13>	Enable protected fields
<-1,14>	Disable protected fields
<-1,15>	Delete line
<-1,16>	Insert line
<-1,17>	Delete character
<-1,18>	Insert character
<-1,19>	Read cursor address
<-1,20>	Read character at current cursor address
<-1,21>	Start blinking field
<-1,22>	End blinking field
<-1,23>	Enable alternate character set
<-1,24>	Disable alternate character set
<-1,25>	Set horizontal position
<-1,26>	Set vertical position
<-1,27>	Set terminal attributes
<-1,28>	Cursor on
<-1,29>	Cursor off

Code	*Purpose*
<-1,30>	Start underscore
<-1,31>	End underscore
<-1,32>	Start reverse video
<-1,33>	End reverse video
<-1,34>	Start reverse blink
<-1,35>	End reverse blink
<-1,36>	Turn off screen display
<-1,37>	Turn on screen display
<-1,38>	Top left corner
<-1,39>	Top right corner
<-1,40>	Bottom left corner
<-1,41>	Bottom right corner
<-1,42>	Top intersection
<-1,43>	Right intersection
<-1,44>	Left intersection
<-1,45>	Bottom intersection
<-1,46>	Horizontal line
<-1,47>	Vertical line
<-1,48>	Intersection
<-1,49>	Solid block
<-1,50>	Slant block
<-1,51>	Cross-hatch block
<-1,52>	Double line horizontal
<-1,53>	Double line vertical
<-1,54>	Send message to function key line
<-1,55>	Send message to shifted function key line
<-1,56>	Set normal display format
<-1,57>	Set horizontal split
<-1,58>	Set vertical split (39 character columns)
<-1,59>	Set vertical split (40 character columns)

Code	*Purpose*
<-1,60>	Set vertical split (column to next character)
<-1,61>	Activate split segment 0
<-1,62>	Activate split segment 1
<-1,63>	Send message to host message field
<-1,64>	Up arrow
<-1,65>	Down arrow
<-1,66>	Raised dot
<-1,67>	End-of-line marker
<-1,68>	Horizontal tab symbol
<-1,69>	Paragraph
<-1,70>	Dagger
<-1,71>	Section
<-1,72>	Cent sign
<-1,73>	One-quarter
<-1,74>	One-half
<-1,75>	Degree
<-1,76>	Trademark
<-1,77>	Copyright
<-1,78>	Registered trademark
<-1,79>	Print screen
<-1,80>	Set to wide (132-column) mode
<-1,81>	Set to normal (80-column) mode
<-1,82>	Enter transparent print mode
<-1,83>	Exit transparent print mode
<-1,84>	Begin writing to alternate page
<-1,85>	End writing to alternate page
<-1,86>	Toggle page
<-1,87>	Copy to alternate page
<-1,88>	Insert column
<-1,89>	Delete column
<-1,90>	Block fill with attribute
<-1,91>	Block fill with character
<-1,92>	Draw a box
<-1,93>	Scroll box up one line
<-1,94>	Scroll box down one line
<-1,95>	Select jump scroll
<-1,96>	Select fast smooth scroll
<-1,97>	Select medium-fast smooth scroll
<-1,98>	Select medium-slow smooth scroll
<-1,99>	Select slow smooth scroll
<-1,100>	Start underscored, blinking field
<-1,101>	End underscored, blinking field
<-1,102>	Start underscored, reverse field
<-1,103>	End underscored, reverse field
<-1,104>	Start underscored, reverse, blinking field
<-1,105>	End underscored, reverse, blinking field
<-1,106>	Start underscored text without space
<-1,107>	End underscored text without space
<-1,108>	Start reverse text without space
<-1,109>	End reverse text without space
<-1,110>	Start reverse blinking text without space
<-1,111>	End reverse blinking text without space
<-1,112>	Start underscored blinking text without space
<-1,113>	End underscored blinking text without space
<-1,114>	Start underscored reverse text without space
<-1,115>	End underscored reverse text without space
<-1,116>	Start underscored reverse blinking text without space
<-1,117>	End underscored reverse blinking text without space
<-1,118>	Start blink without space
<-1,119>	End blink without space
<-1,120>	Set cursor to blinking block
<-1,121>	Set cursor to steady block
<-1,122>	Set cursor to blinking underline
<-1,123>	Set cursor to steady underline
<-1,124>	Reserved
<-1,125>	Reserved

Code	*Purpose*
<-1,126>	Reserved
<-1,127>	Reserved
<-1,128>	Select top status line without address
<-1,129>	End status line (all kinds)
<-1,130>	Select unshifted status line without address
<-1,131>	Select shifted status line without address
<-1,132>	Select black text
<-1,133>	Select white text
<-1,134>	Select blue text
<-1,135>	Select magenta text
<-1,136>	Select red text
<-1,137>	Select yellow text
<-1,138>	Select green text
<-1,139>	Select cyan text
<-1,140>	Select black reverse text
<-1,141>	Select white reverse text
<-1,142>	Select blue reverse text
<-1,143>	Select magenta reverse text
<-1,144>	Select red reverse text
<-1,145>	Select yellow reverse text
<-1,146>	Select green reverse text
<-1,147>	Select cyan reverse text

Appendix C
Glossary

ASCII A commonly used computer alphabet in which letters, numbers, punctuation marks, and nonprintable control characters are represented in seven **bits**, with values from 0 through 127. See Appendix A.

baud rate The speed at which communications take place, expressed in bits per second (bps). Technically speaking, the baud rate refers to the number of signaling changes per second, rather than bits per second. For older modems and other equipment, the baud rate and bps are the same. Newer modems and equipment generally use several channels or other techniques to get more bps per baud, so their "throughput" in bps exceeds their baud rate. However, Autolog uses `baud` for its "speed of serial port" command for historic reasons.

bit The smallest unit of a computer character. A bit can have a value of 0 or 1, also called space and mark, respectively. Most computers group bits into **bytes,** composed of 8 bits. A typical serial communication character is composed of 10 bits: the data byte of 8 bits, plus a start bit and a stop bit that mark the beginning and end of each character. Since the popular computer alphabet **ASCII** uses only 7 bits to represent each character, some computers or modems may use 10-bit serial characters composed of the 7-bit data, a **parity** bit, plus a start and a stop bit.

bps Bits per second. See **baud rate**.

break The reversal of the normal "idle" state of the communications channel. In other words, if the channel is "off" when no characters are coming or going, a break turns the channel "on." (Technically, an idle channel is "on," and is turned "off" during a break condition!) Breaks are used as attention-getting signals or flags for certain functions by some computer systems.

buffer A storage area for data. Your **modem** may have a buffer for storing incoming characters until your computer is ready to read it. Your communications port may have a small buffer (for only one or a few characters) to serve a similar purpose. A **file transfer protocol** is said to be buffered if it can

receive more than one **packet** at a time and hold it until it is acknowledged (unlike an unbuffered protocol, in which no additional packets are sent until the packet last sent is acknowledged).

byte A character composed of eight **bits**.

carrier detect A hardware signal (also called DCD) used by modems and other serial data communication equipment to indicate when a serial communication channel has been established. Your modem is said to have detected carrier (i.e., the DCD signal switches from low to high) when the remote modem answers. Carrier is said to be dropped (i.e., the DCD signal switches from high to low) when the local or remote modem hangs up. The `carrier` command causes Autolog to monitor the carrier detect hardware signal and to return to command mode and abort any file transfer in progress if carrier is lost.

command mode Autolog's state when it is ready to accept your commands or perform a script (`go` file). Command mode is characterized by the Autolog command screen (if Autolog is not in terse mode) and the command prompt >. Characters you type are interpreted by Autolog as commands for it to perform. Press the **change key** to enter **talk mode**.

communications port The hardware that allows your **modem** or other device to connect to your computer. The communications port works in conjunction with interface software to allow your computer to send and receive characters to and from the modem or other device.

cursor The current-position indicator, a character displayed on your computer screen to indicate where your input will be placed when you type on the keyboard. Your cursor may appear as a vertical bar |, a rectangle ■, or an underscore _.

download To receive a copy of a file from the remote system to your local system. Compare with **upload.**

error correction A means of detecting and correcting errors that occur in data transmitted between systems. Some **file transfer protocols** (e.g., SMT, ZMODEM, and YMODEM) provide error correction to ensure that files are transferred correctly between systems. Note that YMODEM-g and XMODEM-g provide error detection but, strictly speaking, not error correction, since they do not provide a means for retransmitting flawed data and will abort if an error is detected. Some modems also provide error correction for *all* data transferred between systems (e.g., data you see on your screen while in talk mode). Modem error correction protocols include LAPM and MNP.

escape sequence A special control sequence of characters that begins with an ASCII escape (ESC) character. Escape sequences are often used as a way to control a device, such as a terminal, at the end of a serial communications channel.

file transfer protocol An agreed-upon method for exchanging files between two computer systems. Popular file transfer protocols include YMODEM and ZMODEM. Both the local and the remote computers must use the same file transfer protocol for a file transfer to succeed, because both computers must agree on the method, timing, and arrangement of the file data and on what to do in the event of an error in the transfer.

hardware flow control A way of preventing data loss using hardware signals between a modem and its communications port. One hardware signal is used to indicate when it's okay to send data from the communications port to the modem (known as "clear to send" or CTS), and another is used to signal when it's okay for the modem to send data to the communications port (known as "request to send" or RTS). The most commonly used connections for this under the RS-232-C wiring standard are pin 4 for RTS (also called RS under the RS-449 standard), and pin 5 for CTS (also called CS under the RS-449 standard). Compare with **software flow control.**

hash code A number used to compare files to determine whether they are identical in content. The contents of a file are scanned and encoded numerically (using one of a variety of algorithms) to obtain the hash code, which might be considered an abbreviated, numeric representation of the file's contents. The hash code of one file can then be compared to the hash code of another (provided both files are "hashed" using the same algorithm) to determine if the contents are identical. Autolog uses an ISO-standard CRC32 hash algorithm for comparing hash codes rather than the hash utilities available under different operating systems, which may differ from each other.

modem A hardware device that transforms data from the form used between your computer and its peripheral devices (such as terminals and printers) into a form that can be transferred across a phone line. Your modem may be *internal* (a card inside the computer) or *external* (a separate device that connects to the computer by a modem cable).

packet A file transfer unit consisting of a chunk of data from the file to be transferred, along with control data for error correction or data compression. The size and composition of the packet depend on the **file transfer protocol** in use. Strictly speaking,

during error-correcting file transfers, packets are exchanged in both directions between the sending and receiving systems. Packets may consist of file data and control information (data packets), may be sent by the receiving system to acknowledge receipt of data packets (acknowledgments), or may contain other special information to coordinate the exchange of files, depending on the protocol in use.

parity A very simple error correction method used by some computer equipment. Equipment that uses parity requires that the parity bit (usually the 8th bit added to a 7-bit character) be set to 0 or 1 depending on the number of other 0 and 1 bits in the character. Autolog's `parity` and `strip` commands are useful when communicating with such equipment.

software flow control A simple means of preventing data loss during high-speed communications using the special characters XON and XOFF in the data stream. Less effective than **hardware flow control**, software flow control may also prevent certain types of file transfers (including XMODEM, YMODEM, and sometimes ZMODEM) from working.

talk mode Often called terminal mode, this is Autolog's state when you are talking to the remote computer or to your modem. The characters you type are sent to the communications port over which the remote connection takes place. Data received over the remote channel are displayed on the screen. Press the **change key** to return to Autolog's **command mode** if you need to give Autolog a command.

upload To send a copy of a file from the local computer to the remote computer. Compare with **download.**

Index