

The Autolog Subroutine for Softworks BASIC and Ashell

| | | |
|----------|-----------------------------------------------------------------|-----------|
| 1 | The Format of the Autolog XCALL..... | 1 |
| 1.1 | MODE..... | 1 |
| 1.2 | DATBYTES..... | 2 |
| 1.3 | ERRCOD..... | 2 |
| 1.4 | TIMER'TICKS, PATTERN'MATCH, MACRO'NO..... | 2 |
| 2 | Mapping Argument Variables..... | 2 |
| 2.1 | MODE Argument..... | 2 |
| 2.2 | DATBYTES Argument..... | 2 |
| 2.2.1 | DATBYTES as a Character String..... | 3 |
| 2.2.2 | Unformatted DATBYTES..... | 3 |
| 2.3 | ERRCOD Argument..... | 6 |
| 2.4 | MACRO'NO Argument..... | 7 |
| 2.5 | TIMER'TICKS Argument..... | 7 |
| 2.6 | PATTERN'MATCH Argument..... | 7 |
| 3 | The Eight Types of Autolog XCALLS..... | 8 |
| 3.1 | Mode 0: Bidirectional Character I/O..... | 8 |
| 3.2 | Mode 1: Command Mode..... | 10 |
| 3.3 | Mode 2: Talk Mode..... | 12 |
| 3.4 | Mode 3: Display Screen..... | 13 |
| 3.5 | Mode 4: Read Data..... | 14 |
| 3.5.1 | The debug input Command..... | 17 |
| 3.6 | Mode 5: Send Data..... | 18 |
| 3.6.1 | The debug output Command..... | 18 |
| 3.7 | Mode 6: Read Macro..... | 18 |
| 3.8 | Mode 7: Interruptible Read Data Mode..... | 19 |
| 4 | Aborting the Autolog Subroutine..... | 20 |
| 5 | Preparing to Run BASIC Programs with Autolog XCALLS..... | 20 |
| 6 | Contents of the BASIC RUN File..... | 21 |

Soft Machines
PO Box 14
Sidney, IL 61877

(217) 688-3317
support@softmach.com

The Autolog Subroutine for Softworks BASIC and Ashell

A BASIC XCALL interface for Autolog is available for Softworks BASIC or Ashell. The Autolog XCALL allows you to perform complex, unattended communications operations controlled from BASIC programs using Autolog. And, unlike the original Alpha Micro version of Autolog, the new XCALLs do not require preloading Autolog before BASIC or RUN is executed.

We assume in this document that you are already familiar with BASIC.

The new XCALL interface automatically executes Autolog as required by the XCALL, and no preloading is required. In fact, you *must not start up Autolog* before your BASIC program that makes XCALLs to Autolog. However, because shared memory and semaphores are used to communicate between Autolog and BASIC, if more than one user at a time are using the Autolog XCALL, you may need to increase the shared memory and semaphore kernel variables to ensure that the XCALL will not abort due to a lack of resources.

1 The Format of the Autolog XCALL

Within your BASIC program that controls Autolog operation, you will need to make XCALLs to the Autolog subroutine. The format of most BASIC XCALLs to the Autolog subroutine is:

```
XCALL AUTLOG, MODE, DATBYTES, ERRCOD, (TIMER'TICKS,) (PATTERN'MATCH)
```

The Autolog XCALL requires at least three arguments: MODE, DATBYTES, and ERRCOD. In addition, for some XCALLs you may include the optional TIMER'TICKS and PATTERN'MATCH arguments. A new mode to read Autolog macro expressions follows this format:

```
XCALL AUTLOG, MODE, MACRO'NO, DATBYTES, ERRCOD
```

The following sections explain each argument (also called *parameters* of the Autolog subroutine) in detail.

1.1 MODE

The MODE argument tells Autolog which of eight types of modes to act in: command mode, one of four different character I/O modes with the remote system, talk mode, or reading one of the 16 Autolog macros. There is also a mode that simply tells the Autolog subroutine to display the Autolog command screen. For instance, when you want to give Autolog an Autolog command, you would make a command mode XCALL. When you want to send data to, or receive data from, the remote system, you would make an XCALL in one of the four character I/O modes: The bidirectional character I/O mode sends data to the remote computer, and when the subroutine returns, it also returns data received from the remote system; the “send data” character output mode simply sends data to the remote system; the “read data” character input mode only receives data from the remote system; there is an interruptible and an uninterruptible type of read data mode. The command screen display mode needs to be used only when you want your BASIC programs to “look like” the normal Autolog display. The “read macro” mode allows you to store the contents of any of the 16 Autolog macros into a BASIC variable. The talk mode XCALL should be used only when designing BASIC programs that will run interactively with a user present, since the Autolog subroutine will return from a talk mode XCALL only when the user presses the **change key**. The mode is indicated by a two-byte binary variable with a value from 0 to 7, as follows:

- 0 **Bidirectional character I/O mode**
- 1 **Command mode**
- 2 **Talk mode**
- 3 **Display screen mode**
- 4 **Read data mode**
- 5 **Send data mode**
- 6 **Read macro mode**
- 7 **Interruptible read data mode**

We will discuss how the XCALL works in each of the different modes in Section 3 below on “The Eight Types of Autolog XCALLs.”

1.2 DATBYTES

The `DATBYTES` argument contains the data to be acted on. For a command mode `XCALL`, the `DATBYTES` argument would contain the Autolog command you want executed (for instance `dial 555-1212` or `transmit xyz.lst`). For the character I/O modes, `DATBYTES` would contain the data you want to send to the remote system, or when the subroutine returns it will contain the data received from the remote system, or it may function in both ways, depending on which of the character I/O modes you use. In talk mode `XCALLs` and “command screen display” `XCALLs`, `DATBYTES` is ignored, although you must always give this argument even for these types of `XCALLs`. The `DATBYTES` variable should be mapped as either a string or as an unformatted variable. The behavior of the Autolog subroutine varies, depending on how this variable is mapped. We discuss the format of the `DATBYTES` variable in more detail in Section 2.2.

1.3 ERRCOD

The `ERRCOD` argument will contain an error status when the subroutine returns. The variable you use for the `ERRCOD` argument must always be a two-byte binary variable, mapped into two parts. One byte of this variable indicates any error that may have occurred, using the numbers 1 to 127 for file system errors, 129 to 255 for Autolog errors, or zero if no error occurred. The other byte is used as a special flag. For `XCALLs` in some modes, it will be zero if the error occurred on the local system or 255 if the error occurred remotely. For “read data” character input mode, this flag is used as a logical boolean status byte that will indicate the status of the received data. The `ERRCOD` argument must always be given for all `XCALLs` to the Autolog subroutine. We will discuss the proper mapping of the `ERRCOD` variable in Section 2.3. How the `ERRCOD` variable is used by each type of `XCALL` is explained in Sections 3.1 through 3.8.

1.4 TIMER'TICKS, PATTERN'MATCH, MACRO'NO

The optional `TIMER'TICKS` and `PATTERN'MATCH` arguments are used only by “read data” `XCALLs`. You can include them in other `XCALLs`, but they will be ignored. We will look at these arguments in more detail later in Sections 2.5 and 2.6. Briefly, `TIMER'TICKS` will indicate a maximum length of time to wait for data from the remote system, and `PATTERN'MATCH` will contain an expected prompt or response from the remote system that you want to wait for.

`MACRO'NO` is used only by “read macro” `XCALLs`. It is used to indicate which of the 16 Autolog macros you want to read. It is discussed in Section 2.4

2 Mapping Argument Variables

The variables you use as arguments to the Autolog `XCALL` should all be mapped variables to ensure that they are of the proper size and type. You can give these variables whatever names you like; however, in this manual we'll stick to the variable names given as examples in the following sections.

2.1 MODE Argument

The variable you use to specify the `MODE` argument should always be mapped as a two-byte binary variable. You can give this variable any name you want, but we will refer to this variable by the name **MODE**.

The `MODE` variable should be mapped as follows:

```
MAP1 MODE, B, 2
```

The `MODE` variable should always have a value of 0 through 7 (in decimal). A discussion of the eight types of modes follows in Section 3 below.

2.2 DATBYTES Argument

The `DATBYTES` argument is used to pass data between BASIC and Autolog. For modes that send data, the contents of `DATBYTES` are sent either to Autolog in command mode or to the remote system (as if in talk mode). `DATBYTES` is also used to store received characters for modes that receive data. `DATBYTES` is ignored in modes that don't exchange data, such as mode 3 (display the Autolog command screen), although the `DATBYTES` argument must be included in all `XCALLs` to the Autolog subroutine, regardless of mode.

How `DATBYTES` functions depends on the mode being used in an `XCALL`. Before you make a mode 0 (bidirectional I/O) or a mode 5 (send data) `XCALL`, you place the data that you want to send to the remote system in `DATBYTES`. When a mode 0 (bidirectional I/O), mode 4 (read data), or mode 7 (interruptible read

data) XCALL returns, the data received from the remote system will be in the DATBYTES variable. Similarly, when making mode 1 (command mode) XCALLs, DATBYTES must contain the Autolog command you wish to execute. Mode 6 (read macro) will return the contents of an Autolog macro in the DATBYTES argument. Modes 2 and 3 do use data in the DATBYTES variable, although the XCALL statement itself must include the DATBYTES argument.

The variable you use for the DATBYTES argument can have any name you want, but we will use the name **DATBYTES** to refer to this variable. DATBYTES should always be a mapped variable and can be mapped as either a string or an unformatted variable. The information that can be passed and how the subroutine behaves depend on whether DATBYTES is a string variable or an unformatted variable. We'll look at each type of DATBYTES variable in the following sections.

☞ Due to the new way that Autolog and BASIC communicate, the maximum size of a string or unformatted DATBYTES variable is 3000 bytes. If you need to use a buffer larger than this, then the very first XCALL in your BASIC session must use the largest buffer size that you will require for the entire session. For example, to use a buffer of 30,000 bytes, your first XCALL in the session would need to use a mapped variable whose size is 30,000 bytes (plus two bytes for the binary character count for unformatted variables):

```
! Example of forcing buffer size above 3000 characters.
MAP1 DATBYTES
  MAP2 CHAR' COUNT, B, 2
  MAP2 BUFFER, S, 30000
...
MODE=1
BUFFER="LINK MODEM1"
CHAR' COUNT=LEN (BUFFER)
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
```

Using this method, you can allocate buffers up to 65,534 bytes in length.

2.2.1 DATBYTES as a Character String

For most applications, you can declare the DATBYTES variable to be a regular string variable. DATBYTES as a string is very easy to map and use, but the types of data that you can send or receive are limited by the nature of string variables.

The DATBYTES should always be a mapped variable, such as:

```
MAP1 DATBYTES, S, 80
```

The length of the DATBYTES string is up to you. However, you should make sure that it is large enough to pass all the data that you'll need to the remote system and, most important, large enough to store a reasonable amount of data received from the remote system without losing any.

DATBYTES as a string is easy to use because you do not have to declare its size before each XCALL, as you do with an unformatted DATBYTES variable. However, you will not be able to send or receive certain data bytes with the string type of DATBYTES. Nulls pose a major drawback when using a string variable. Since a null is used to indicate the end of a string variable, you will not be able to send or receive nulls to or from the remote system if your DATBYTES variable is a string. Also, bytes with the eighth bit set may cause problems if DATBYTES is a string variable.

2.2.2 Unformatted DATBYTES

DATBYTES can be mapped as a specially defined unformatted variable. The unformatted DATBYTES allows you to send and receive all binary characters, including nulls (binary zero) and values with the eighth data bit set. (BASIC sometimes clears the eighth data bit in each character of a string variable. Also, BASIC defines the end of a string as a null byte, which prevents the use of binary zeroes as data.) If you need to send or receive nulls or non-ASCII characters with the eighth bit set, you will need to use an unformatted DATBYTES variable.

The unformatted DATBYTES must consist of two parts: a two-byte binary variable used as a count of the number of incoming/outgoing data bytes and a variable used as the data storage area. Here is a MAP statement example to make the arrangement clearer:

```

MAP1 DATBYTES
    MAP2 CHAR' COUNT, B, 2      ! character count
    MAP2 TEXT' AREA, X, 1000   ! data area

```

Figure 1 Simple Unformatted DATBYTES Map

Here the variable `DATBYTES` is divided into two `MAP2`-level variables, which we will call **CHAR' COUNT** and **TEXT' AREA** (although you may give these variables any names you wish). The first level 2 variable *must* be a two-byte binary variable. The second level 2 variable can be of any size you wish and should be an unformatted variable. You can also use additional overlay maps for other types of variables but, for the moment, we will assume that the second level 2 variable, `TEXT' AREA`, is an unformatted variable only.

When the `DATBYTES` variable is an unformatted variable, the Autolog subroutine will always use the first two bytes as a “character count.” This is why, for ease of use, an unformatted `DATBYTES` variable should be mapped into two parts. When the `DATBYTES` variable is an unformatted variable mapped into two parts, as in Figure 1, the Autolog subroutine will use the `CHAR' COUNT` variable to store the size (number of bytes) of the data stored in the `TEXT' AREA` variable. Before you make an `XCALL` in a mode that sends data (mode 0, mode 1, or mode 5), you must set `CHAR' COUNT` equal to the size of the data in `TEXT' AREA` (`TEXT' AREA` will contain the data that you want to send to the remote system or the Autolog command you want to execute). When an `XCALL` that reads data returns (mode 0, mode 4, mode 6, or mode 7), the received data will be in the `TEXT' AREA` variable, and the number of bytes received will be in the `CHAR' COUNT` variable. Modes 2 and 3 do use or place data in the `DATBYTES` variable, although the `XCALL` statement itself must include the `DATBYTES` argument.

This manipulation of the `CHAR' COUNT` and `TEXT' AREA` variables is very important when making `XCALLS`. If you should forget to set `CHAR' COUNT` to the proper size when sending data, the Autolog subroutine will probably not send the data that you want sent. If, for instance, you want to send the command “dir” and a carriage return to the remote system, and the text has been put into the `TEXT' AREA` variable, but you forget to set `CHAR' COUNT` to the proper size of 4 bytes, either too many or too few characters may get sent to the remote system, depending on the previous value of `CHAR' COUNT`. When using an unformatted `DATBYTES` variable, any change to the `TEXT' AREA` part of `DATBYTES` should always be accompanied by a change to the `CHAR' COUNT` part of `DATBYTES`.

Using an unformatted `DATBYTES` variable also gives you a great deal of flexibility in the format of the text variable. The following map introduces some new variable formats that you can use. This group of map statements allows you to access the incoming or outgoing data as a string, as a binary byte array, or as an unformatted variable.

```

MAP1 DATBYTES
    MAP2 CHAR' COUNT, B, 2      ! character count
    MAP2 TEXT' AREA, X, 1000   ! unformatted data area OR
    MAP2 AS' ARRAY (1000), B, 1, @TEXT' AREA ! data as an array of bytes OR
    MAP2 AS' A' STRING, S, 1000, @TEXT' AREA ! data area as a string

```

Figure 2 Unformatted DATBYTES Map with Overlays

☞ The “@`TEXT' AREA`” portion of the `AS' ARRAY` and `AS' A' STRING` maps allows you to save memory by using the same space for three different variables. BASIC allows you to “overlay” variables as long as you won’t need to use those variables at the same time. Since we need only one of the variables `TEXT' AREA`, `AS' ARRAY`, or `AS' A' STRING` at a time, we save 2000 bytes by overlaying `AS' ARRAY` and `AS' A' STRING` in the same space as `TEXT' AREA`. This also means that no matter which variable we use to store our text, the data area begins in the proper “place”: after the two-byte `CHAR' COUNT` area.

Before an `XCALL` that sends data (mode 0, 1, or 5), `CHAR' COUNT` must be set to the number of characters to be output, and those characters must be loaded into either the `TEXT' AREA`, `AS' ARRAY`, or `AS' A' STRING` variable. Note that even when you are treating your data as a string by using the `AS' A' STRING` variable, you must set `CHAR' COUNT` to the length of the string if you `XCALL` Autolog using the variable `DATBYTES`. Remember that the information about variables that a BASIC `XCALL` will pass to an external subroutine is

limited to the size, location, and format of the variable. If the variable that you use for the DATBYTES argument is unformatted, the Autolog subroutine will *always* look in the first two bytes to find the number of bytes to send. For example, assuming that you use the maps in Figure 2, if you make the following XCALL:

```
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
```

Autolog knows only that the variable "DATBYTES" is an unformatted variable at a particular location. It will therefore always look at the first two bytes of DATBYTES for the size of the data to send. It will then look at the area following those initial two bytes, count off the specified number of bytes and send them. However, if you use the following XCALL:

```
XCALL AUTLOG, MODE, AS'A' STRING, ERRCOD
```

Autolog only knows that the variable "AS'A' STRING" is a string variable either of a specified maximum length or that ends in a null if it is less than the maximum length. In this case, the Autolog subroutine will not look at the CHAR' COUNT variable at all and will simply send the string contained in AS'A' STRING. If these distinctions seem unclear, please review the *AlphaBASIC User's Manual*, in particular the chapters on variable types, memory mapping, and the XCALL statement.

When an XCALL that receives data (mode 0, 4, 6, or 7) returns, CHAR' COUNT will contain the number of data characters received from the communications channel, and the data will be available in the unformatted TEXT' AREA variable. When an XCALL that receives data is called using an unformatted DATBYTES variable, the first two bytes will always contain the number of bytes of data that were received, and the received data will always be stored as an unformatted variable in the area just past the first two bytes of DATBYTES.

As an example of using an unformatted DATBYTES variable, here is a subprogram to send a six-byte escape sequence to the remote computer.

```
! BASIC routine to send a six-character sequence as an array of
! binary bytes, using the maps from Figure 2.
AS' ARRAY(1)=155 ! ESC with highest bit set
AS' ARRAY(2)=255 ! TAB(-1,
AS' ARRAY(3)=0 ! 0)
AS' ARRAY(4)=155 ! ESC with highest bit set
AS' ARRAY(5)=12 ! ROW 12
AS' ARRAY(6)=40 ! COLUMN 40
CHAR' COUNT=6
MODE = 5
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
```

The unformatted variable type is used here to allow sending binary characters (155 and 0), which are not normally permitted in BASIC strings. [The six characters above represent the string sequences the AUTGEN terminal driver uses to send CRT functions TAB(-1,0) and TAB(12,40).] The unformatted type of DATBYTES variable must be used whenever binary data of this type is to be received or sent. Using different format variable overlays gives you the flexibility to choose the easiest format for the kinds of data you'll be sending or receiving. Using the same overlay maps, we can also send and receive data as a simple string. For example:

```
! BASIC routine to send Autolog the command "GENERIC TRUE" using an
! unformatted DATBYTES variable, using the maps from Figure 2.
AS'A' STRING="GENERIC TRUE"
CHAR' COUNT=LEN(AS'A' STRING)
MODE=1
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
```

For this simple XCALL, using the same maps as before, we could also use the following routine:

```
! BASIC routine to send Autolog the command "GENERIC TRUE" using a
! string variable, using either the maps from Figure 2 or a simple
! string-type variable.
AS'A' STRING="GENERIC TRUE"
MODE=1
XCALL AUTLOG, MODE, AS'A' STRING, ERRCOD
```

Notice the difference in the XCALL statement between this example and the preceding one: In the first example we used the unformatted DATBYTES variable when we made the XCALL, and so we had to set CHAR' COUNT to the proper length of our string. In the second example, we used the string variable AS'A' STRING in the place of the DATBYTES argument, so no CHAR' COUNT is needed. Use care in your use of variables: When you make an XCALL make sure that the variable you use for the DATBYTES argument is of the appropriate format.

2.3 ERRCOD Argument

ERRCOD is a mapped two-byte binary variable that returns a zero for normal execution or an error code for abnormal execution. If ERRCOD is zero, the XCALL was performed correctly; if it is nonzero, the XCALL was terminated due to an error.

The ERRCOD returned has two parts: the command error code, contained in the lower byte of ERRCOD, and a special purpose flag mapped to the upper byte of ERRCOD. They may be separated using the following map:

```
MAP1 ERRCOD
      MAP2 AUTERR, B, 1
      MAP2 REMERR, B, 1
```

Figure 3 ERRCOD Map

You should check the status of AUTERR after each XCALL is performed. A value of zero indicates that the command executed without error. Values from 1 to 127 represent operating system error codes. Values from 129 to 255 represent Autolog errors. **The new Ashell XCALL returns error code 158 when a FINISH or QUIT command is given.** You will find a list of all other error codes in Chapter 4 of the *Autolog User's Guide*, "Automating Autolog with Script Files."

REMERR is a flag used differently by each mode type of XCALL. For most modes, the REMERR flag is used to indicate whether the error occurred on the local system or the remote system. When AUTERR is a nonzero value, indicating that an error has occurred, REMERR will be set to 255 if the error occurred on the remote system or to zero if the error occurred on the local system. You need to test REMERR only when AUTERR is nonzero after XCALLs in most modes. However, mode 4 (read data) XCALLs use the REMERR byte as a flag to indicate the status of the incoming data buffer. The use of REMERR by mode 4 XCALLs is discussed in more detail in Section 3.5.

2.4 MACRO'NO Argument

The MACRO' NO argument is used only by mode 6 (read macro expression) XCALLs, and is used to indicate which of Autolog's 16 macros you want to read. The MACRO' NO argument may be mapped as either a floating point or binary number, as in:

```
MAP1 MACRO' NO, F, 6
      or
MAP1 MACRO' NO, B, 2
```

The variable MACRO' NO should be set to a number from 0 to 15 to indicate which Autolog macro expression (\$0-\$F) you want to read. (The contents of the specified macro will be returned in the DATBYTES variable.)

2.5 TIMER'TICKS Argument

The `TIMER'TICKS` argument is used only by mode 4 and 7 (read data) XCALLs and is optional. `TIMER'TICKS` is used to specify the amount of time a mode 4 XCALL will wait before returning. If an expected prompt or response from the remote system is specified in the `PATTERN'MATCH` argument, `TIMER'TICKS` specifies the *maximum* amount of time the mode 4 or 7 XCALL will wait before returning (in this case it may return earlier if a match is found).

The maximum allowable value of `TIMER'TICKS` is 2,147,483,647 system timer ticks. System timer ticks are 1/10,000 of a second, or 100 microseconds. `TIMER'TICKS` can range from 0 to 2,147,483,647, or from 0 seconds to over 59 hours.

If you use the `TIMER'TICKS` argument, the variable used should be mapped as either a binary or a floating point number. If a binary variable is used, a size of 4 bytes is sufficient to allow the maximum size permitted for `TIMER'TICKS`. `TIMER'TICKS` could be mapped as:

```
MAP1 TIMER'TICKS, F, 6
      OR
MAP1 TIMER'TICKS, B, 4
```

When used with mode 4, the `TIMER'TICKS` are not interruptible. But if you use mode 7, a user can press the **cancel key** to interrupt `TIMER'TICKS` before it reaches zero. This can be convenient for debugging.

2.6 PATTERN'MATCH Argument

The `PATTERN'MATCH` argument is used only by modes 4 and 7 (read data) XCALLs and is optional. `PATTERN'MATCH` is used to specify an expected prompt or response from the remote system that you wish to wait for when reading data from the remote system.

The variable used for the `PATTERN'MATCH` argument can be either a string or an unformatted variable. If an unformatted variable is used, the first two bytes will be used as a character count indicating the size of the data, just like the convention for the `DATBYTES` argument. Please see Section 2.2 above on the `DATBYTES` argument for information on how `PATTERN'MATCH` can be mapped.

The `PATTERN'MATCH` variable is limited to a maximum of 3000 bytes.

If an unformatted variable is used for `PATTERN'MATCH`, remember that it should be separated into two `MAP2`-level variables, with distinct names to distinguish them from the `DATBYTES` `MAP2` variables. For instance, if you use map statements such as the ones in Figure 2 for `DATBYTES`, you could use the following map statement for an unformatted `PATTERN'MATCH`:

```
MAP1 PATTERN'MATCH
MAP2 PATTERN'SIZE, B, 2    ! character count
MAP2 MATCH'AREA, X, 1000 ! unformatted match area OR
MAP2 MATCH'ARRAY(1000), B, 1, @MATCH'AREA    ! match as array of bytes
MAP2 MATCH'STRING, S, 1000, @MATCH'AREA      ! OR match as a string
```

Figure 4 Unformatted PATTERN'MATCH Map with Overlays

Further discussion on how `PATTERN'MATCH` is used in mode 4 and 7 XCALLs can be found in Section 3.5 on mode 4 XCALLs.

3 The Eight Types of Autolog XCALLs

When making an Autolog XCALL from a BASIC program, the first argument you must give is the *mode*. The mode tells the Autolog subroutine whether the information that follows is to be acted on as an Autolog command or passed to the remote system, or whether the Autolog subroutine should receive data from the remote system or perform another function such as entering talk mode or simply displaying the Autolog screen. You will give the `MODE` argument a value of 0, 1, 2, 3, 4, 5, 6, or 7: These values correspond to the eight different functions you can request, as follows:

- 0 **Bidirectional character I/O mode** allows the BASIC program to send data to and receive data from the remote channel. When `MODE` is equal to 0, the characters contained in the variable `DATBYTES` are passed to the remote computer as if in talk mode. Then all accumulated data that is received from the remote

system is returned to the BASIC program in the DATBYTES argument. However, mode 4 and mode 5 offer an easier method to send and receive characters from the remote system. Mode 0 is still supported only for compatibility with earlier versions of the Autolog subroutine.

- 1 **Command mode** allows the BASIC program to execute most Autolog commands. The command is passed in the variable DATBYTES as it would be entered manually in Autolog (except that you do not need to append a carriage return at the end of the command).
- 2 **Talk mode** invokes the familiar Autolog talk mode as a subroutine call. Control will return to the BASIC program *only* when the user presses the currently defined **change key**. Mode 2 is useful only for attended or interactive communications, since there is no way to return to the BASIC program except by a user's keypress. Mode 2 allows you to build your own "custom" communications package around Autolog.
- 3 **Display screen mode** displays the standard Autolog command screen on the user's CRT. Mode 3 is useful for customized data communications when you want your BASIC application to "look like" Autolog. Note that although the screen is "painted" with the Autolog command screen, commands must still be passed to Autolog in mode 1.
- 4 **Read data mode** allows you to receive characters from the remote system and store them in the DATBYTES variable. You may use the optional TIMER' TICKS and PATTERN' MATCH arguments to set a maximum time for the capture of remote characters and/or to wait for a particular character or set of characters to be received. While mode 4 is waiting for data, the **cancel key** is not honored.
- 5 **Send data mode** allows you to send characters to the remote system. The characters you wish to send should be placed in the DATBYTES variable.
- 6 **Read macro mode** allows you to read the contents of one of the 16 Autolog macros (indicated by the MACRO'NO argument) and stores the contents in the DATBYTES variable. The format of the XCALL statement for mode 6 is slightly different from the format used by other modes.
- 7 **Interruptible read data mode** works just like mode 4, except when waiting for data, it can be interrupted by pressing the **cancel key**.

Each of the following sections discusses each type of XCALL in detail.

3.1 Mode 0: Bidirectional Character I/O

Mode 0 is a bidirectional character I/O call. Modes 4 and 5 are easier to use because they separate the "input" and "output" functions, so we recommend that you use modes 4 and 5 rather than mode 0 in all new programs. Mode 0 is supported only for compatibility with existing programs written for older versions of the Autolog subroutine. Please feel free to skip this section!

When a mode 0 XCALL is made, first all of the data bytes in the DATBYTES variable are sent to the remote site. Then all data received from the remote system (or all that will fit in DATBYTES) is returned in DATBYTES. If no characters are available from the remote site, then DATBYTES will have a length of zero.

Because this mode operates in *real time* and returns *immediately*, it should be called *frequently*. Otherwise, you may lose incoming data due to overflows of Autolog's holding buffer area. As characters are received from the remote system, they are temporarily stored in Autolog's holding buffer. When this buffer fills up, characters are discarded to make room for more. If you don't make frequent XCALLs to capture the incoming data out of Autolog's buffer, some data may be lost.

Also note that because this mode returns immediately, it may contain no data, or perhaps only a portion of the data you need to receive. If you wish to record the complete response of the remote computer to a command sent using an XCALL, you can use a BASIC subprogram that will repeatedly XCALL Autolog and accumulate the entire response into a larger string variable. Each time the subprogram repeats the XCALL, it should empty the DATBYTES variable, XCALL Autolog, then append the returned contents of DATBYTES to the larger string variable being used to record the complete response. The subprogram could be designed to repeat the Autolog XCALL until the AMOS prompt is returned in the DATBYTES variable (a dot "." character) and a certain amount of time elapses in which no new characters are returned. The larger string variable would then contain the complete response from the remote computer. The following example shows how such a subprogram might work:

```

! BASIC routine to capture a response from remote system and store it
! in the string variable RESPONSE. (DATBYTES is a string variable in
! this example.)

RESPONSE=""
MODE=0
X=1          ! X marks the location just past the last "."
T1=86401     ! T1 stores the time the last "." was received
GET' RESPONSE:
    DATBYTES=""
    XCALL AUTLOG, MODE, DATBYTES, ERRCOD
    RESPONSE=RESPONSE+DATBYTES
    IF DATBYTES<>"" THEN GOTO GET' RESPONSE      ! as long as we keep
                                                ! getting new data
T2=TIME      ! T2 is current time if no new data has been rec'd
Y=INSTR(X,RESPONSE, ".")
IF Y<>0 THEN X=Y+1 : T1=TIME          ! start "timer" if we got "."
IF LEN(RESPONSE)>X-1 THEN T1=86401    ! reset timer if more data
IF T2-T1<15 THEN GOTO GET' RESPONSE

GOT' IT:
! Now the complete response is stored in RESPONSE

```

It is possible for mode 0 XCALLs to report certain errors in the ERRCOD variable when the subroutine returns. However, only normally reportable Autolog errors (such as *no remote TRMDEF*) will be reported via ERRCOD. File errors or system errors reported by the remote system will have to be captured via the DATBYTES variable. For instance, if you send the command **COPY *.* = DSK2:IT.TXT [200,0]** to the remote system via a mode 0 XCALL, and the remote system responds with the message *?File not found*, no error will be reported via ERRCOD. This error message would have to be captured via the DATBYTES variable. ERRCOD will only report whether the XCALL was completed without error, and not whether the command or data being sent by the XCALL was completed without error.

The **TIMER' TICKS** and **PATTERN' MATCH** arguments, if included in a mode 0 XCALL, will be ignored.

The debug input and debug output commands, discussed in Sections 3.5.1 and 3.6.1, can be useful for debugging programs that use mode 0 XCALLs or for applications that require the display of information sent and received by a mode 0 XCALL.

Because modes 4 and 5 for character I/O are easier to use, we suggest that you use them rather than mode 0. We have left mode 0 unchanged so that Autolog subroutine users can run vintage BASIC programs without modification.

3.2 Mode 1: Command Mode

Mode 1, command mode, allows Autolog commands to be entered via BASIC. All commands except the **chain** command are allowed in mode 1. The Autolog **chain** command, which is incompatible with BASIC's memory-use scheme, will return the error code for invalid command if attempted.

The **finish** (or **quit**) command is used to end an XCALL session. **With Ashell, the finish command returns an autolog error code of 158.** All other commands function normally. You can dial, set communications options and make adjustments, transfer files—and have BASIC monitor and control these functions.

The Autolog command itself that is to be executed is passed via the DATBYTES argument. A carriage return should not be included in the DATBYTES variable, because a mode 1 XCALL does not require a carriage return at the end of the command being passed.

You may use the same colon commands as can be used in script files to control whether commands, responses, and error messages appear on the screen as Autolog commands are processed. The colon commands that Autolog recognizes are discussed in Chapter 4 of the *Autolog User's Guide*. To use a colon command, set the DATBYTES variable equal to the string *":letter"*, for instance *":S"* to silence command and response display.

Mode 1 XCALLs will report any errors in the ERRCOD variable. The low byte (the AUTERR variable) will be set to a nonzero value if an error occurred or will be zero if no error occurred. The upper byte (the REMERR variable) will be set to 255 if the error occurred on the remote system or will be zero if the error occurred on the local system.

The TIMER/TICKS and PATTERN/MATCH arguments, if included, will be ignored.

Here is an example of a mode 1 XCALL used to send Autolog the dial command. Notice that we check the status of AUTERR after the XCALL to determine if the call was successful, or if not, to determine why it failed.

```
! BASIC routine to dial a phone number, check if the call was completed, !
! and print status of dialing attempt. (DATBYTES is a string variable in !
! this example).
MODE=1
DATBYTES="DIAL 555-1212"
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
IF AUTERR=0 THEN GOTO DIALED'OK
IF AUTERR=238 THEN PRINT "CALL ABORTED" : GOTO DONE
IF AUTERR=239 THEN PRINT "NO ANSWER" : GOTO DONE
IF AUTERR=240 THEN PRINT "LINE IS BUSY" : GOTO DONE
IF AUTERR=241 THEN PRINT "NO DIAL TONE" : GOTO DONE
IF AUTERR=243 THEN PRINT "HAVEN'T SET UP MODEM YET" : GOTO DONE
PRINT "CALL COULD NOT BE COMPLETED - UNSPECIFIED ERROR"
GOTO DONE
DIALED'OK:
    PRINT "CALL COMPLETED AS DIALED"
DONE:
! We're done dialing
```

Here's another sample mode 1 XCALL routine, in which we also use the REMERR part of the ERRCOD variable to determine if an error was local or remote:

```
! BASIC routine to RECEIVE a file and check error status.
! DATBYTES is a string variable, although this routine can easily be
! modified to use an unformatted DATBYTES.
MODE=1
DATBYTES="NOERASE TRUE" ! Prevent accidental erasure of files.
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
IF AUTERR<>0 THEN GOTO TROUBLE ! Quit if command failed
MODE=1
DATBYTES="RECEIVE NEWFIL.TXT=OLDFIL.TXT"
XCALL AUTLOG, MODE, DATBYTES, ERRCOD ! RECEIVE a file
IF AUTERR<>0 THEN GOTO TRANSFER'FAILED
STATUS$="FILE TRANSFER COMPLETED"
GOTO DONE

TRANSFER'FAILED:
    IF AUTERR>155 THEN GOTO TROUBLE ! Error of more than 155
    ! indicates an Autolog command error rather than file error.
    IF REMERR=0 THEN STATUS$="LOCAL " ELSE STATUS$="REMOTE "
    ! Check to see if file error was local or remote.
    ON AUTERR GOTO ERR1, ANYERR, ERR3, ERR4, ERR5, ERR6, ERR7, ERR8
    ! See Autolog User's Guide for list of error codes.
ANYERR:
    STATUS$=STATUS$+"FILE ERROR - UNSPECIFIED"
    GOTO DONE
ERR1:
```

```
STATUS$=STATUS$+"FILE SPEC ERROR"  
GOTO DONE  
ERR3:  
STATUS$=STATUS$+"FILE NOT FOUND"  
GOTO DONE  
ERR4:  
STATUS$=STATUS$+"FILE ALREADY EXISTS"  
GOTO DONE  
ERR5:  
STATUS$=STATUS$+"DEVICE NOT READY"  
GOTO DONE  
ERR6:  
STATUS$=STATUS$+"DEVICE FULL"  
GOTO DONE  
ERR7:  
STATUS$=STATUS$+"DEVICE ERROR"  
GOTO DONE  
ERR8:  
STATUS$=STATUS$+"DEVICE IN USE"  
GOTO DONE  
  
TROUBLE:  
STATUS$=DATBYTES+" COMMAND FAILED"  
  
DONE:  
PRINT STATUS$
```

3.3 Mode 2: Talk Mode

Mode 2 allows you to enter Autolog talk mode. Once you have entered talk mode, *your keypresses* are sent to the remote site and remote data is displayed on your screen, just like talk mode in regular Autolog use. This mode continues until the change key is pressed or loss of carrier occurs (if the `carrier` option is turned on and your hardware supports this feature). This means that once a mode 2 XCALL has been made, the subroutine *will not return* until a user presses the **change key** or a loss of carrier is detected. Mode 2 should therefore never be used in BASIC programs that are to run without an operator present. Mode 2 should only be used when designing a custom user interface or when writing a BASIC program that will be run interactively with an operator present.

You must include the `MODE`, `DATBYTES`, and `ERRCOD` arguments in a mode 2 XCALL. However, the `DATBYTES` argument is not used by a mode 2 XCALL, and any data in the `DATBYTES` variable will be ignored. If `TIMER' TICKS` and `PATTERN' MATCH` arguments are included, they will also be ignored.

Any `get` or `send` file that is active will be available using the **copy key**. (You must use mode 1 to execute a `get` or `send` command.)

You can use talk mode when you want to let the user use the remote system “hands on” after you have automated the dialing and log-on functions using `command` and `input/output` mode XCALLs. Remember that mode 2 (talk) requires a user to press the **change key** before the BASIC program will continue to execute.

Here’s a sample of a BASIC routine using a mode 2 XCALL:

```

! BASIC routine to let user enter talk mode.
MODE=2
DATBYTES="" ! DATBYTES will be ignored, but nonetheless we must give
! all three arguments
PRINT "YOU ARE NOW GOING TO ENTER TALK MODE"
PRINT "YOU WILL BE TALKING DIRECTLY WITH THE REMOTE SITE"
PRINT "PRESS THE CHANGE KEY WHEN DONE"
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
! During this interlude, the user will be in talk mode, & can perform
! whatever tasks he/she wishes on the remote system. The Autolog
! subroutine will not return until the user presses the CHANGE key.
PRINT "BASIC PROGRAM EXECUTION WILL NOW RESUME"

```

Mode 2 XCALLs will report any normally reportable Autolog error via the ERRCOD argument. Generally only Autolog errors such as *no remote TRMDEF* will be reported in the AUTERR variable. System or file errors that occur on the remote system while the user is in talk mode *will not* be reported in the ERRCOD argument; therefore the user should monitor activity on the remote system to make sure the commands that he or she enters have executed on the remote system. The AUTERR variable will indicate only any error that prevented the XCALL from working, that is, errors that prevent Autolog from entering talk mode.

3.4 Mode 3: Display Screen

Mode 3 paints the Autolog command display on your screen. This mode is intended primarily for the programmer who wants to create a custom Autolog environment, yet retain the traditional Autolog command screen with little effort. Note that even when the screen has been painted with the command screen, commands must be passed to Autolog using mode 1 XCALLs.

Although all three required arguments (MODE, DATBYTES, and ERRCOD) must be included when making a mode 3 XCALL, all except the MODE variable will be ignored. If a TIMER' TICKS or PATTERN' MATCH argument is included, it will be ignored also.

Here is a sample BASIC routine using a mode 3 XCALL to make a custom user interface for Autolog. We've also thrown in a mode 1 (command mode) and mode 2 (talk mode) XCALL just to give you an idea of how such a custom interface could work.

```

! BASIC routine illustrating a customized user interface
MODE=2
DATBYTES="" ! DATBYTES will be ignored, but nonetheless we must give
! all three arguments
PRINT "YOU ARE NOW ENTERING TALK MODE"
PRINT "YOU WILL BE TALKING DIRECTLY WITH THE REMOTE SITE"
PRINT "PRESS THE CHANGE KEY WHEN DONE"
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
! During this interlude, the user will be in talk mode, & can perform
! whatever tasks he/she wishes on the remote system. The Autolog
! subroutine will not return until the user presses the change key.
MODE=3
DATBYTES="" ! DATBYTES will be ignored, but nonetheless we must give
! all three arguments
XCALL AUTLOG, MODE, DATBYTES, ERRCOD ! Autolog command screen
! is now displayed
INPUT "ENTER AUTOLOG COMMAND: ",DATBYTES ! Prompt for command
MODE=1
XCALL AUTLOG, MODE, DATBYTES, ERRCOD ! Execute command

```

3.5 Mode 4: Read Data

Mode 4 allows you to capture characters sent by the remote system. In addition, by using the optional `TIMER' TICKS` and `PATTERN' MATCH` arguments, you can wait for an expected response to be received before the subroutine will return. Unlike mode 0 (bidirectional I/O), this XCALL is *one way*, that is, it is used to *receive characters only*.

When the subroutine returns, the characters received from the remote system will be stored in the `DATBYTES` variable. Note that if the `DATBYTES` variable is not cleared out before making a mode 4 XCALL (by setting `DATBYTES` to a null string if a string variable is used or by setting `CHAR' COUNT` to zero if an unformatted variable is used), any data received will be *appended* to the `DATBYTES` variable. If a string variable is used, the new characters received will be appended to the end of the string. If an unformatted variable is used, the number of new characters received will be added to `CHAR' COUNT`, and the new characters received will be placed after the existing bytes (as indicated by `CHAR' COUNT`) in the unformatted area. (See Section 2.2 above for information on the `DATBYTES` variable.) This feature provides automatic buffering and allows you to store a complete response within the `DATBYTES` variable, even between several mode 4 XCALLs, provided of course that the variable is large enough to store the complete response.

To make mode 4 XCALLs even easier and more useful, you can also include the optional `TIMER' TICKS` and `PATTERN' MATCH` arguments. `TIMER' TICKS` is used to set the maximum amount of time that you wish the XCALL to wait before returning. `PATTERN' MATCH` allows you to specify an expected response or sequence of characters that you wish to wait for.

`TIMER' TICKS` measures time using system timer ticks as a unit. There are 10,000 timer ticks per second. `TIMER' TICKS` can range from 0 to 2,147,483,647, or from 0 to over 59 hours. The minimum nonzero amount of time you can specify is 1 timer tick, or 100 microseconds. However, the accuracy of the timer depends on the system, and the waiting time may be longer than 100 microseconds. If `TIMER' TICKS` is zero or is not included in the XCALL, the subroutine returns immediately (with whatever characters may have been on hand returned in the `DATBYTES` variable). If `TIMER' TICKS` is nonzero, but no `PATTERN' MATCH` was included in the XCALL, the subroutine will wait the amount of time specified by `TIMER' TICKS` (while gathering characters from the remote) and then return. If a `PATTERN' MATCH` is included in the XCALL, the subroutine may return sooner if the `PATTERN' MATCH` is received. If the `PATTERN' MATCH` is not received after the amount of time specified by `TIMER' TICKS` has elapsed, the subroutine will return, and a special flag will be set in the `REMERR` byte of the `ERRCOD`. (We'll discuss the use of the `ERRCOD` argument in a moment.)

The `PATTERN' MATCH` variable, if included in a mode 4 XCALL, should contain an expected prompt, response, byte, or sequence of bytes that you wish to wait for the remote system to send. The subroutine will return immediately if a complete byte-for-byte match has been received. While the subroutine is waiting for a match, the characters received are buffered and the count of incoming characters (`CHAR' COUNT` if an unformatted variable is used) is constantly updated.

If `TIMER' TICKS` and `PATTERN' MATCH` are included in a mode 4 XCALL, the subroutine's job will `sleep` for 1000 timer ticks or the remaining `TIMER' TICKS` count, whichever is less. The 1000-tick delay is the length of time it would take for about 100 characters to be sent at 9600 baud. Then it will "wake up" and check to see if the bytes contained in `PATTERN' MATCH` have been received by scanning for them in Autolog's internal buffer area. If they are found, the subroutine will return. If not, and if there is still remaining time in the `TIMER' TICKS` count, the subroutine will `sleep` again for 1000 timer ticks or the remaining `TIMER' TICKS` count, whichever is less, and the cycle repeats until either the `PATTERN' MATCH` is found or the amount of time specified by `TIMER' TICKS` elapses.

If a match occurs, `TIMER' TICKS` will be set to the value of `TIMER' TICKS` at the time the match was found, and this value is guaranteed to be nonzero if a match occurs.

There is another event that will cause the subroutine to return prematurely. The subroutine will return, even if there is time left in `TIMER' TICKS` and no match is received, *if the DATBYTES variable becomes full and cannot accept more data*. When the subroutine returns, a flag will be set in the `REMERR` variable, and `TIMER' TICKS` will be nonzero.

In order to efficiently report the possible conditions when the subroutine returns, a mode 4 XCALL makes special use of the `ERRCOD` variable. If the subroutine returns without finding a match, if the `DATBYTES` variable was filled up, or if the subroutine "times out," the `AUTERR` variable (lower byte of `ERRCOD`) will be set to 156. The `REMERR` variable (upper byte of `ERRCOD`) will contain a flag indicating which of the following condition(s) occurred:

- 2 – TIMER' TICKS time elapsed.
- 4 – No match received.
- 8 – DATBYTES variable full, no more room to store received data.

Since it is possible for more than one of these conditions to be present when the subroutine returns, the REMERR variable will contain the sum of various error flags that occurred. For instance, if the subroutine “times out” without finding a match, REMERR will be set to 6: the sum of 2 (time elapsed) and 4 (no match received). If the DATBYTES variable becomes full, and the subroutine had to return before a match was found, REMERR would equal 12: the sum of 4 (no match received) and 8 (DATBYTES variable full). If, by coincidence, a match is found just as the DATBYTES variable fills up, REMERR would be set to 8, indicating that your variable is full (and should now be cleared), but that the match you were waiting for was received.

When the subroutine returns, there are several variables you can check to see what the status is. TIMER' TICKS will be zero if a “time out” occurred. TIMER' TICKS is guaranteed to be nonzero if a match was found, but it also may be nonzero if the subroutine returned early due to a “filled up” DATBYTES variable. Checking the AUTERR variable will show if any of the error flags are set: AUTERR will be zero if no flags are set or 156 if any of the flags are set. REMERR will give specific information on what error conditions occurred. You can decide how much or how little information is needed for each XCALL and examine the error indicators accordingly.

Here are some example XCALLs. All are in an acceptable format.

```
MODE=4
STRING=""
XCALL AUTLOG, MODE, STRING, ERRCOD

MODE=4
CHAR' COUNT=0
TIMER' TICKS=10000
XCALL AUTLOG, MODE, DATBYTES, ERRCOD, TIMER' TICKS

MODE=4
STRING=""
TIMER' TICKS=2345678
PATTERN' MATCH="OK"+CHR$(13)+CHR$(10)
XCALL AUTLOG, MODE, STRING, ERRCOD, TIMER' TICKS, PATTERN' MATCH
```

As you can see, the TIMER' TICKS and PATTERN' MATCH arguments are optional and need not be included. (However, if you want to specify a PATTERN' MATCH, you must always include TIMER' TICKS also.) You can specify TIMER' TICKS without a PATTERN' MATCH (as in the middle example above) but be aware that, assuming no other errors occur (such as the DATBYTES variable filling up), a “time out” will always occur so REMERR will always be 2 when the subroutine returns.

In the examples above, we “emptied out” our DATBYTES variable before calling the subroutine. This is not required, and often you may wish to leave existing characters in the DATBYTES variable so that you can buffer up a response. Here is a sample routine that will take advantage of this buffering feature to wait for *x* lines of data terminated by linefeeds, or until 45 seconds elapse. The complete *x* lines received will be put into the DATBYTES variable (assuming this variable is large enough to store them all).


```

! BASIC routine to capture X number of lines from the remote system and !
store them in the unformatted variable DATBYTES, or "bomb out" if more !
than 45 seconds elapse without getting all the lines. The variable X
! should be defined before calling this subprogram.
SET'UP:
  MODE=4
  CHAR'COUNT=0                ! empty out DATBYTES variable
  TIMER'TICKS=45 * 10000       ! wait 45 seconds maximum
  PATTERN'MATCH=CHR$(10)      ! match on linefeed
NEXT'MATCH:
  IF X=0 GOTO DONE
  XCALL AUTLOG,MODE,DATBYTES,ERRCOD,TIMER'TICKS,PATTERN'MATCH
  ! When the subroutine receives a linefeed, it will return. CHAR'COUNT
  ! is increased by the number of new bytes received, and TIMER'TICKS
  ! will be decreased by the number of seconds it took to receive the
  ! linefeed.
  IF AUTERR=0 THEN X=X-1 : GOTO NEXT'MATCH
  IF AUTERR#156 THEN ? "OTHER ERROR (E.G., NO REMOTE TRMDEF)":GOTO DONE
  FIND'ERROR=REMERR/2
  ON FIND'ERROR GOSUB ERR1,ERR2,ERR3,ERR4,ERR5,ERR6,ERR7
DONE:
  PRINT TEXT'AREA[1,CHAR'COUNT]
  RETURN

! Error code subprograms start here
ERR1:
  PRINT "TIMED OUT - NOT ALL LINES RECEIVED"
  RETURN
ERR2:
  PRINT "DIDN'T FIND THE END OF THAT LAST LINE"
  RETURN
ERR3:
  GOSUB ERR2 : GOSUB ERR1 : RETURN
ERR4:
  PRINT "YOU NEED A BIGGER VARIABLE FOR THIS!"
  RETURN
ERR5:
  GOSUB ERR4 : GOSUB ERR1 : RETURN
ERR6:
  GOSUB ERR4 : GOSUB ERR2 : RETURN
ERR7:
  GOSUB ERR4 : GOSUB ERR3 : RETURN

```

3.5.1 The debug input Command

The `debug input` option can be useful for debugging mode 4 XCALLs or for applications that require the display of information received by a mode 4 XCALL. Normally, the characters received by a mode 4 XCALL are not displayed on the screen. However, if you're having trouble with a BASIC program that uses mode 4 XCALLs, it may be informative to see the characters as they are received from the remote system.

To have the received characters displayed on the screen, turn on the `debug input` option. This is done by passing the command string **debug input true** to the Autolog subroutine via a mode 1 (command) XCALL.

3.6 Mode 5: Send Data

Mode 5 lets you send characters to the remote system. Unlike mode 0 (bidirectional character I/O mode), this XCALL works *one way*: to *send characters only*. To receive the remote system's responses to the characters you send, you would perform a mode 4 XCALL.

A mode 5 XCALL requires the `MODE`, `DATBYTES`, and `ERRCOD` arguments. If `TIMER' TICKS` and `PATTERN' MATCH` are also included, they will be ignored.

The `DATBYTES` variable should contain the characters that you want sent to the remote system. When the XCALL is made, the contents of the `DATBYTES` variable are sent to the remote system, and the subroutine returns immediately. The `DATBYTES` variable will remain *unchanged*. The only errors that will be reported in the `ERRCOD` variable will be errors that prevented the Autolog subroutine from transmitting the data, such as *no remote TRMDEF*. These errors would be reported in the lower byte of `ERRCOD` (the `AUTERR` variable).

Here is a sample mode 5 XCALL to send the remote system the `sysstat` command:

```
! Sample BASIC routine to send a command to the remote system.
! DATBYTES is a string variable in this example.
MODE=5
DATBYTES="SYSTAT"+CHR$(13)
XCALL AUTLOG, MODE, DATBYTES, ERRCOD
```

3.6.1 The debug output Command

The `debug output` option can be useful for debugging mode 5 XCALLs or for applications that require the display of information sent by a mode 5 XCALL. Normally, the characters sent by a mode 5 XCALL are not displayed on the screen. However, if you're having trouble with a BASIC program that uses mode 5 XCALLs, it may be informative to see the characters as they are sent to the remote system.

To have the transmitted characters displayed on the screen, turn on the `debug output` option. This is done by passing the command string `debug output true` to the Autolog subroutine via a mode 1 (command) XCALL.

3.7 Mode 6: Read Macro

Mode 6 lets you read the contents of any of Autolog's 16 macro expressions. The contents of the specified macro expression are returned in the `DATBYTES` argument. More information about Autolog macros and how they can be assigned can be found in Chapter 4 of the *Autolog User's Guide*.

A mode 6 XCALL requires the `MODE`, `MACRO' NO`, `DATBYTES`, and `ERRCOD` arguments. The format of a mode 6 XCALL is slightly different from the other XCALL modes. When making a mode 6 XCALL, the format should be:

```
XCALL AUTLOG MODE, MACRO' NO, DATBYTES, ERRCOD
```

The `MACRO' NO` should be set to a number from 0 to 15 to indicate which Autolog macro (`$0-$9`, `$A-$F`) you want to read. When the XCALL returns, the contents of the specified macro are returned in the `DATBYTES` variable. In general, the only errors that will be reported in the `ERRCOD` variable will be syntax errors, such as when you specify an illegal macro number like "16." These errors would be reported in the lower byte of `ERRCOD` (the `AUTERR` variable).

Below is a sample mode 6 XCALL to read all 16 Autolog macros and display them on the screen.

```
! Sample BASIC routine to send a command to the remote system.
! DATBYTES is a string variable in this example.
MODE=6
FOR X = 1 TO 16
  MACRO' NO = X - 1
  XCALL AUTLOG, MODE, MACRO' NO, DATBYTES, ERRCOD
  PRINT "THE MACRO ";MACRO' NO;" IS: <";DATBYTES;">"
NEXT X
```

The display generated by this example will be something like this:

```
THE MACRO 0 IS: <contents of $0>
THE MACRO 1 IS: <contents of $1>
THE MACRO 2 IS: <contents of $2>
THE MACRO 3 IS: <contents of $3>
THE MACRO 4 IS: <contents of $4>
THE MACRO 5 IS: <contents of $5>
THE MACRO 6 IS: <contents of $6>
THE MACRO 7 IS: <contents of $7>
THE MACRO 8 IS: <contents of $8>
THE MACRO 9 IS: <contents of $9>
THE MACRO 10 IS: <contents of $A>
THE MACRO 11 IS: <contents of $B>
THE MACRO 12 IS: <contents of $C>
THE MACRO 13 IS: <contents of $D>
THE MACRO 14 IS: <contents of $E>
THE MACRO 15 IS: <contents of $F>
```

3.8 Mode 7: Interruptible Read Data Mode

Mode 7, interruptible read data mode, works just like mode 4, read data mode, except that a user can interrupt mode 7, or force it to return prematurely, by pressing the **cancel key**. Mode 4 ignores the **cancel key** and returns only on the conditions previously described in Section 3.5.

4 Aborting the Autolog Subroutine

The Autolog subroutine aborts if the XCALL contained an improper number of arguments. (The Ashell version aborts to the AMOS-like shell, the Softworks BASIC version aborts to the Unix shell.) You will get the following message if this occurs:

```
? Number of arguments
```

The Autolog subroutine always requires at least three arguments: MODE, DATBYTES, and ERRCOD. This is true even of XCALLs (such as mode 2, talk, or mode 3, display screen) that may not use all three arguments. It is up to you to ensure that the arguments given are of the proper type. The TIMER' TICKS and PATTERN' MATCH arguments may be included in *any* XCALL but will be ignored by all but mode 4 or mode 7 (read data) XCALLs. TIMER' TICKS and PATTERN' MATCH are never required, not even for mode 4 or mode 7 XCALLs. Mode 6 XCALLs require four arguments: MODE, MACRO' NO, DATBYTES, and ERRCOD.

The Autolog subroutine will also abort with an error if there is a mismatch between the size of the variable used as an argument and the required size of the argument. For instance, the MODE argument must be a two-byte binary variable. If you use a three-byte variable for the mode argument, you will get this error when you try to run the BASIC program:

```
? arg #1 in error - too large
```

You will also get an error if the variable used is too small. For example, if you use a one-byte binary variable for the ERRCOD argument (which should be a two-byte binary variable), you will get the error

```
? arg #3 in error - too small
```

If the variable type of the variable actually used is not the required type for a certain argument, you will get the error message

```
? arg #{num} in error - type mismatch in AUTLOG.SBR
```

where *{num}* indicates which argument had the wrong type (1 indicates the first argument; 2, the second argument; and so on). For instance, if you use a variable mapped as a string for the TIMER' TICKS argument (which must be either a binary or floating point variable) in a mode 4 XCALL, you would get the error

```
? arg #4 in error - type mismatch
```

The Autolog subroutine will *not* abort if various unforeseen mishaps occur, such as if the remote system crashes in the middle of the communications session, or the phone lines go dead, or the connection is cut off in the middle of a communications session. Therefore, if your BASIC interface to Autolog is to operate truly

unattended, some foresight should be used in programming to take into account various accidents that can occur. With a little thought and planning, however, a careful programmer can construct a powerful automatic BASIC interface program with the Autolog subroutine that can take advantage of the ability to create unattended communications sessions (and also take advantage of late-night phone rates!).

5 Preparing to Run BASIC Programs with Autolog XCALLs

No preparation is required before running a BASIC program that contains AUTLOG XCALLs, unlike the AMOS version of the Autolog subroutine.

However, make sure that your program always hangs up the modem. When using Ashell, if your program is aborted by control-C, the Autolog session may continue in the background. *Always* make sure you issue a `hangup`, or `finish` command anytime you have placed a call. The `finish` command hangs up the modem only if you have used the `dial` command: If you dial using your own AT commands, you are responsible for hanging up the modem as well.

If your Ashell program terminates prematurely, before a `finish` was given to Autolog, you can use the following simple program or a similar one to make sure the modem is hung up and Autolog is finished properly.

```
! Simple program to finish Autolog after a premature program abort.
MAP1 MODE,B,2,1
MAP1 AUTERR,B,2
XCALL AUTLOG MODE,"FINISH",AUTERR
```

6 Contents of the BASIC RUN File

The BASIC RUN file XCALLs Autolog to perform the required commands, or converse with the remote system.

The `link` command can be used to designate the port used for communications. After the `link` command is invoked, the `ERRCOD` argument will be set to zero if the `link` was performed. Otherwise, `ERRCOD` will note why the `link` failed.

The `modem` command can be specified from BASIC as well, if you wish. The Autolog subroutine returns an `ERRCOD` of zero if the `modem` command worked or a nonzero value if there was a problem.

You can also use the `go` command as an XCALL from the BASIC program. Note that when the Autolog subroutine returns to the main program, the `ERRCOD` indicates only the status of the last Autolog command that executed, and *not* whether any particular command in the script file failed. For instance, your BASIC program could make an XCALL that sends Autolog the command `go setup`, where `setup.go` contains a `link` and a `modem` command to establish your communications channel. If the `go setup` command fails (e.g., if `setup.go` cannot be located), the `ERRCOD` will indicate why the `go` command failed. If the `go` command executes without problem, the Autolog subroutine will execute the commands contained in `setup.go` and return. However, the `ERRCOD` argument will indicate only whether the last command in the script file executed.

Once your communications channel and modem type have been acquired and established, you may use the `dial` command to autodial. The `ERRCOD` argument will return with a value of zero if the remote computer's modem has answered and a connection has been established. Other error codes will be returned for "no dial tone," "line is busy," etc. Note that your modem must be capable of reporting these conditions for these error codes to be returned. See Table 4.1 in the *Autolog User's Guide* for notes on dialing errors.

Once you are connected, you can use the `say` command or the character input and output modes 0, 4, 5, and 7 to send data bytes to the remote computer. Using the character input/output modes, you can receive data bytes from the remote computer, interpret them, and generally have more control over how data is passed to and from the remote system. You may choose to use the `debug` command to have input to and output from the remote system displayed on the screen as it is received or sent.

You may also use the file transfer functions to transfer files between the two systems, provided the remote system has the proper support software for these functions.

When you are through communicating with the remote site, you can use the `hangup` command to hang up the modem.

The last XCALL to the Autolog subroutine should be a `finish` command. You need not `chain`—indeed, you cannot `chain`—to the Autolog program to delete memory modules, as required in the AMOS version. The

`finish` command is all that is needed to properly end Autolog subroutine use with Softworks BASIC and Ashell. Unlike under AMOS, the finish command returns error code 158 when the Autolog client has ended.