AutoMate ™ Communications Driver

User's Manual

Version 2.625 --- October 21, 1996

<u>SOFTWARE LICENSE AGREEMENT</u>

IMPORTANT! The enclosed materials are provided to you on the express condition that you agree to this Software License. By opening the diskette envelope or using any of the enclosed diskette(s) you agree to the following provisions. If you do not agree with these license provisions, return these materials to Automation Consulting Services, Inc., in original packaging with seals unbroken, within 3 days from receipt, for a refund.

1. This software and the diskette on which it is contained (the "Licensed Software"), is licensed to you, the end user, for your own internal use. You do not obtain title to the Licensed Software or any copyrights or proprietary rights in the Licensed Software. You may not transfer, sub-license, rent, lease, convey, copy, modify, translate, convert to another programming language, decompile, or disassemble the Licensed Software for any purpose.

2. The Licensed Software is provided "as-is". All warranties and representations of any kind with regard to the Licensed Software are hereby disclaimed, including the implied warranties of merchantability and fitness for a particular purpose. Under no circumstances will the Manufacturer or Developer of the Licensed Software be liable for any consequential, incidental, special, or exemplary damages even if apprised of the likelihood of such damages occurring. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Incorporated Driver Amendment

If you own the ACS AutoMate BASIC driver (Incorporated Version), this license is amended to provide for the free or for-profit distribution of software incorporating BASIC Driver code as follows: you may distribute executable programs containing the complete and unaltered ACS AutoMate BASIC Driver (Incorporated Version). The Incorporated Version Libraries may not be copied, sold, modified, distributed, or used by more than one user at a time; they are treated as Licensed Software as described above. You can only distribute the Driver as a part of self-standing executable code (EXE files). No royalties or additional licenses are required to distribute such standalone programs.

For Windows DLLs, you may distribute the DLL (distribution) version without royalties, but you may *not* distribute the Development (VBX) version. It is treated as Licensed Software as described above.

Table of Contents

New in Version 2.5

There have been several important changes in the operation of the AutoMate™ Communications Driver since the last version. If you are updating from a previous version of the Driver, please read this section carefully to ensure proper operation.

Resident Driver

The Resident driver has been discontinued. The complete functionality of the Driver is now contained in what was formerly the binding in previous versions. For example, for Quick Basic version 4.5, the whole Driver is contained in the Quick Libraries.

There is no longer any resident portion of the Driver to load at the command line. In the case of Quick Basic, starting QB with the normal command line:

```
    QB program /L QBDRV
```

will load the driver automatically. There is no longer any need to load the QBDRV.EXE file from the DOS command line before starting your application; indeed, the new version does not use an executable file.

SentinelC Key

In order to improve reliability, we have changed to a new type of hardware key manufactured by Rainbow Technologies. Unlike previous keys, this key can be used on parallel ports other than LPT1. To change the port where the Driver will look for the key, call BASDRV with the function synopsis:

```
    CALL BASDRV(B_KEYPORT, STATUS, PORTN, B, B, B)
```

where PORTN is the integer number of the parallel port where the key is located (1 to 3).

If you are using the default port, LPT1, there is no need to call the KEYPORT function. Rather than checking the key once upon loadup, the Driver now checks the key at random intervals. The Driver will return a status of -2 if the key is not detected.

The KEYPORT command is described in more detail on page 99.

Support for “software key” protection has been discontinued.

R-Net PC Link

Beginning with Version 2.6, the Driver supports the Reliance R-Net PC Link card, which provides direct communications over R-Net without needing a Gateway.

New in Version 2.0

Version 2.0 is a complete rewrite of the AutoMate^{tm} Driver that introduces a number of new features, including:

- Interrupt-driven communications to increase reliability even under multitasking operating environments like DesqView™ and Microsoft Windows 3.0.
- More thorough error checking on caller parameters.
- Support for COM3 and COM4, even on systems that do not automatically detect these ports.
- Support for Microsoft Windows™ version 3.0 or later via the AutoMate Communications Driver DLL.

Warning! Version 2.0 of the Driver uses your computer's hardware interrupts to ensure reliable communications. It is **ABSOLUTELY ESSENTIAL** that you allow the Driver to deactivate these interrupts before you exit your application program. If you fail to do this, your computer may crash unpredictably, even long after you have left your program.

Please see the OFF command, page 94, for more information.

Introduction

The BASDRV AutoMate Communications Driver is a utility that allows programs to communicate with Reliance Electric AutoMate programmable controllers.

The driver provides an easy way for the user to develop programs that access an AutoMate processor's points, registers, and application memory. Information is passed to and from the processor via normal integer and string variables. The BASIC "CALL" statement provides access to the communication functions.

BASDRV communicates with the AutoMate processor via your PC's serial port or over the R-Net PC Link Board.

Using the Driver

The Driver comes in two different "flavors," memory-resident and linkable. To use the Resident Driver, you load it once from the DOS command level. After the driver is in memory, it will remain there until you reset or turn off the computer. The driver consumes about 8 K of user memory. Beginning with Version 2.5, only Turbo Pascal uses a Resident driver.

Other languages, such as C and QuickBASIC, use a "linkable" Driver that becomes part of your application program. A program that uses a linkable Driver has no TSR (memory resident) portion to load. To run an application that uses the linkable Driver, you need only attach the copy protection device to the parallel port and run the application normally.

There is a special linkable version of the Driver, called the "Incorporated Driver," which is intended for commercial developers and users who will need to run applications that use the Driver on many computers. Application programs created with the Incorporated Driver can be distributed without royalties, though the Driver libraries remain under a single-user license.

The AutoMate™ driver is available with interfaces to a variety of languages, including IBM Interpreted BASIC, Microsoft QuickBASIC, Microsoft C, and Turbo PASCAL. The individual interfaces are described at the end of this section, just before the function synopses.

The Windows Driver DLL

ACS provides a version of the Driver that runs under the Microsoft Windows™ operating environment. This Driver is automatically loaded by your application at runtime. The Driver DLL provides two caller interfaces, one suitable for Visual BASIC (and other languages that require the PASCAL calling convention), and one used for languages that support the C calling convention. Both interfaces are found in the same DLL.

General Information

All of the language interfaces share the same general operating principles.

For the Resident driver, there is memory-resident portion of the driver that must be loaded before you can access the AutoMate device. You must load this resident portion directly from PC-DOS. Once it has been loaded, it will remain resident and available until you turn off or reset your computer.

This is especially important for languages like Turbo Pascal and Interpreted BASIC that provide a complete operating “environment.” For these languages, you must always load the Driver before loading the language environment.

The individual language interfaces provide the necessary facilities to invoke the resident portion of the driver. Usually, the language interface is a small object file or code segment that simply routes your commands and data to and from the Driver’s resident portion.

The Linkable or Incorporated Driver is loaded or linked directly into your application program. It does not require any resident portion. Each language has its own method for loading external modules; consult the relevant section (or your language’s documentation) for more information.

“Include” files

Many languages, such as C and Turbo PASCAL, have facilities for “include” or “unit” files that can establish constants and function definitions. Wherever possible, ACS has supplied appropriate include files to make programming easier.

Sample Files

Each version of the Driver is supplied with one or more demonstration programs. Please take the time to examine and run these demonstrators; a few minutes with the samples can save you a lot of frustration. Since the sample programs are known to run, you can use them to test your hardware setup. If the demonstration programs won’t run, your own code probably won’t either. Also, since we wrote the samples, it will be easy for us to diagnose problems encountered while working with them.

The sample programs can give you a head start on your own application by showing you proven ways to construct an application program. In fact, you may wish to simply “cannibalize” the demonstrator programs to fit your own application.

Help !

If you have trouble, have any questions about how the driver works, or want advice about special applications, please be sure to contact us... a two minute phone call could save you hours of frustration. We are more than willing to help you use any *unmodified* software provided by ACS. We will also answer questions about your programs (and help you debug programs that use BASDRV) as time permits.

If you find a bug in BASDRV, be sure to let us know. To help us fix the bug, document it as completely as possible. If you are not sure whether the bug is in your program or the driver, please ship us your program on a PC-compatible disk with documentation of the problem. If the problem is in the driver, we will locate and repair it and return your disk as quickly as we can.

Please read the function descriptions carefully before you use them to save yourself time and trouble later. **Be sure to save your program frequently!** Always save it before you run it the first time. If you have made a mistake (such as a BASIC BASDRV call with less than six parameters), the computer may crash and have to be reset. Unfortunately, BASIC makes absolutely no provision for the type of error checking that would prevent these problems, so you have to be very careful.

Good luck!

Copy Protection

Unfortunately, software piracy is a problem that plagues all program developers: the temptation to copy an unprotected disk is great, and there is little actual danger to the pirate. But copy protection often offends users and sometimes involves unnecessary “hassles”. In order to keep everyone honest with a minimum of trouble for the user, ACS has decided to issue all of its single-user Driver products in copy-protected form.

Note. Incorporated versions of the Driver are not copy protected.

Hardware Lock

Programs protected with a Hardware Lock come on ordinary floppy diskettes. You can (and should) make a backup copy of the protected files, using the DOS DISKCOPY command if you wish. The protection is incorporated into the files themselves and into the locking device.

The Hardware Lock itself is a small device resembling a “gender changer”. It has two 25-pin connectors on it, one male and one female.

When you run a program protected with a Hardware Lock, the software will examine your computer’s parallel printer port. If the correct Hardware Lock is found, the program runs normally. If the locking device is not present, the Driver will return an error code.

To use the Hardware Lock, simply copy the original program diskettes into a directory on your hard disk. Next, plug the male end of the Hardware Lock device into your computer’s parallel printer port (LPT1). If there is a printer already attached to your system, simply plug its cable into the female end of the Hardware Lock.

Once you have attached the locking device, you are ready to run the software. Your computer should operate just as before; the device is only active when the software specifically queries it. The Lock is also transparent to printing.

By default, the Driver looks for the Hardware Key on printer port LPT1. To change the port where the Driver will look for the key, call BASDRV with the function synopsis:

```
CALL BASDRV(B_KEYPORT, STATUS, PORTN, B, B, B)
```

where PORTN is the integer number of the parallel port where the key is located (1 to 3). See the description of the KEYPORT command below (page 99) for more details.

If you are using the default port, LPT1, there is no need to call the KEYPORT function. The Driver will return a status of -2 if the key is not detected.

Cabling

Normally, your ACS software will be supplied with a cable suitable for connecting the PC to the AutoMate processor, Gateway, or Serial Communications Card.

However, some of our customers find that they need to make their own cables. This section describes the cable and pinouts at each end of the connection. The serial port pinouts are included for reference, since they are not often described in computer manuals.

PC Serial Port

The PC serial port is a DB25M (25-pin Male) connector. Here are its pinouts (pins not marked are No Connection):

Pin	Direction	Signal
1	Shield Ground	
2	Output	Transmit Data
3	Input	Receive Data
4	Output	Request to Send
5	Input	Clear to Send
6	Input	Data Set Ready
7	Signal	Ground
8	Input	Carrier Detect
9	Output+	Transmit Current Loop
11	Output-	Transmit Current Loop
18	Input+	Receive Current Loop
20	Output	Data Terminal Ready
22	Input	Ring Indicator
25	Input-	Receive Current Loop

Note: Only strictly IBM-compatible serial ports implement the 20ma current loop interface.

AT Serial Port

The PC AT serial port is a DB9M (9-pin Male) connector. Here are its pinouts:

Pin	**Direction**	**Signal**
1	Input	Carrier Detect
2	Input	Receive Data
3	Output	Transmit Data
4	Output	Data Terminal Ready
5		Ground
6	Input	Data Set Ready
7	Output	Request to Send
8	Input	Clear to Send
9	Input	Ring Indicator

The Cable

You can use the Driver with a three-wire (Transmit Data, Receive Data, and Ground) cable. ACS uses the following cable:

Conductor	**Signal**	**PC Pin**	**AT Pin**	**AutoMate Pin**
1	Ground	7	5	7
2	TD	2	3	3
3	RD	3	2	2

Interfaces

Interpreted BASIC

The Resident driver is provided in a file called BASDRV.EXE. No Incorporated Version is available for Interpreted BASIC. To load the driver into memory, type:

```
BASDRV     r
```

at the PC-DOS command level. The driver will load, display its signon banner, and return to PC-DOS.

Once the driver is resident in memory, you can enter and leave BASIC as many times as you wish. You do not have to reload the driver unless you reset the computer.

When you load the driver, it stores the segment number where it resides in the inter-application communication area reserved by DOS. The segment number can be found at address 0:4F2.

Once you have loaded BASIC, you will need the segment number to call the driver. To get it, you should include the following statements in you program:

```
DEF SEG = 0[ Sets user segment to 0 ]
BASSEG = PEEK(&H4F2)+PEEK(&H4F3)*256  [ Read BASDRV seg. ]
DEF SEG = BASSEG[ Establish access to driver ]
BASDRV = 0[ Call to offset 0 within segment ]
```

These statements determine the location of the driver by reading it from address 0:4F2 and then set up for calls to the driver. Assigning 0 to BASDRV merely sets the destination address within the segment to 0.

After you have executed these statements, you can call the driver with a BASIC CALL statement of the form:

```
CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)
```

where FNO is the function number, STATUS is the return code, and ARG1 through ARG4 are parameters.

YOU MUST ALWAYS CALL THE DRIVER WITH SIX PARAMETERS. The BASIC machine language interface is not very flexible; the only straightforward way to make the large number driver functions readily accessible is to use a fixed length parameter block and a series of function codes.

The CALL statement always expects variable parameters. There is no way to “call by value”. Therefore, you must always assign constant parameters to a variable before executing the CALL. Also, you should be very careful about the types of variables in the CALL. There is no way for the driver to make sure that you have passed the right number and type of parameters, so you may disrupt your system if you make a mistake. In fact, the most common type of problem that you will experience will probably be bizarre driver behavior due to incorrect parameter types or number.

You should also be sure that variables are defined before you pass them to the driver. BASIC sometimes “hangs up” if you pass an undefined parameter to a machine language subroutine.

Since nearly all of the parameters required by BASDRV are integers, you may wish to use the DEFINT statement at the beginning of your program to define one or more letters as “default” integers. If you do so, you can use variables beginning with those letters to communicate with the driver. This method will avoid many annoying and difficult to trace problems.

The first parameter in the BASDRV CALL statement (FNO in the above example) is always the function number. This integer number selects the driver function that you wish to use. There are about fifty standard functions provided by the driver.

The second parameter (STATUS in the sample call) is used to communicate error conditions. Any nonzero value indicates a problem occurred during the transfer. See the table of error values for more complete information. You should check the status codes after each call.

The third, fourth, fifth, and sixth parameters vary according to the function being called.

Sample Files

Two BASIC program files are provided on the distribution diskette to help you use BASDRV. They are named BDHEADER.BAS and BDSHORT.BAS. BDHEADER.BAS contains a commented header that takes care of setup for BASDRV calls. It declares variables beginning with ‘B’ to be integers and assigns function numbers to named variables to make calling BASDRV functions easier. For example, function number 29 (WHORU, read identifying information) is assigned to the variable B.WHORU.

BDHEADER.BAS also performs the standard call to establish communications.

BDSHORT.BAS is a version of BDHEADER.BAS which has all of the comments removed.

We suggest that you start with one of the header files when developing programs the use BASDRV. Doing so will help you avoid some of the more annoying bugs that you might otherwise encounter.

There is another program file, called BDEMO.BAS, on the distribution diskette. This program is intended as a sample of what can be done with BASDRV. Intended for the A20/A30/A40, it provides a set of functions for AutoMate program testing. To use it, issue the following commands at DOS command level:

```
BASDRV    r
BASIC BDEMO    r
```

It may take several seconds for the program to display anything. Once the program is running, its operation is more or less self-explanatory.

BDEMO should provide a good introduction to BASDRV’s capabilities. Feel free to adapt or modify BDEMO if you wish. We recommend that you at least look over BDEMO’s program listing to get a better feel for how the driver operates.

QuickBASIC V4.5

Both Incorporated and Single-User Linkable versions of the Driver are available for Microsoft QuickBASIC Version 4.5.

The QuickBASIC 4.5 Driver is new (and completely incompatible) with previous versions of the QuickBASIC Driver.

Important Note! The QuickBASIC 4.5 Driver is *not* compatible with the BASIC interpreter, as past versions were, nor is it compatible with QuickBASIC prior to V4.5. This Driver can ONLY be used with QuickBASIC Version 4.5 or later.

When entering QuickBASIC, you must load the Driver when you load QB in order to be able to call it. To do this, make sure that the Quick Library file QBDRV.QLB is in the current directory. Then type:

```
QB <BASIC file> /L QBDRV    r
```

This command loads QuickBASIC with the Driver resident. If you do not load QBDRV, you will get "Unresolved Subprogram Reference" errors because the Driver is not loaded.

If you are using a Single-User Linkable version of the Driver, the software will randomly check for the presence of the Hardware Key on the parallel port. If the key is not detected, the Driver will return a STATUS of -2.

If you did not load the resident portion of the Driver into memory before running QuickBASIC, the binding will return a STATUS of -2.

To load the driver and QuickBASIC with the supplied QuickBASIC demo program QBDEMO, type the following command:

```
QB QBDEMO /L QBDRV    r
```

Note that the file BASDRV.BI (the Driver Include file), found on the distribution disk, must be in the current directory before you try to compile the demo.

Press f + % to run the demo. Make sure that you have connected the communications cable to the AutoMate before starting the program.

Converting from the Interpreter

If you are converting a program to QB from the BASIC Interpreter version of the Driver, you will have to make certain changes to your programs. When using the interpreted BASIC binding, you included the following instructions:

```
DEF SEG = 0
BASSEG = PEEK(&H4F2) + PEEK(&H4F3)*256
DEF SEG = BASSEG
BASDRV = 0
IF BASSEG = 0 THEN PRINT"Driver not installed.":END
```

These instructions are not needed when using the QuickBASIC binding, and should not be included. In fact, if you do include them, QuickBASIC may crash. The QuickBASIC binding performs the equivalent of these operations automatically on the first CALL BASDRV instruction. However, you should check for a return status of -2 in case the Driver resident portion was not loaded.

Constant Values

QuickBASIC version 4.5 will automatically create "variables" to contain numeric constants in function calls. This eliminates the irksome process of assigning values to variables, then passing them in the function call. For example, you can set the destination node to 3 with the command:

```
CALL BASDRV(B_SETNOD, STATUS, 3, B, B, B)
```

There is no need to assign 3 to a variable before calling the Driver; QuickBASIC will create "pseudovariables" automatically that contain the constant.

MEMUSE : Memory Usage

In order to support the AutoMate 40, which can have more than 32K of memory, the Driver MEMUSE uses longword integers. In QuickBASIC, use the DIM … AS LONG statement to create a longword array. The QBDEMO program has a sample MEMUSE call that you may wish to examine.

DECLARE & Header file

For your convenience, we have included a "Header" include file which defines all of the QBDRV function numbers as CONSTants. This file is called BASDRV.BI. You may wish to use the $INCLUDE metacommand to bring this file into your BASIC Driver programs.

WARNING! *Do Not* use the old (Driver version 1.2 or earlier) include file with the Version 2.0 Driver. This will cause your program to crash!

The Include file also includes the statements:

```
DECLARE SUB BASDRV CDECL (BYVAL NF AS INTEGER, SEG ST AS
  INTEGER, SEG P1 AS ANY, SEG P2 AS ANY, SEG P3 AS ANY, SEG P4
  AS ANY)

DECLARE SUB BASDRVOFF CDECL ()
```

These statements ***must*** be included in all QB programs that call the Driver, especially if you are converting from an older version of the Driver. It will cause QuickBASIC to send segmented addresses to the Driver, in addition to making sure that you always call BASDRV with the proper number of parameters.

If you are upgrading from a previous version of the Driver, make sure that you are using the new version of the Include file, or that the declarations in your program look like those shown above. If you use the old declarations with the new Driver, your program is guaranteed to crash!

Making EXE files

You will need the QBDRV.LIB file (supplied with the driver) when you go to create EXE files using the QuickBASIC “Make EXE” command (Run menu), or if you want to compile and link from the DOS command line. This file should be in the current directory when you invoke the Make EXE command; QB will automatically incorporate the code into the output EXE file.

Incorporated Version

If you are using the Incorporated Version of the Driver, most of this section still applies to you, with one important exception: the file names and load procedures are slightly different. The Incorporated Driver comes in two files, QB4INC.QLB and QB4INC.LIB. These Libraries contain the complete Driver machine code for direct incorporation into your application programs.

The Incorporated Version QB command is:

```
      QB *file* /L QB4INC    r
```

This command will load QuickBASIC with the QB4INC.QLB library resident. If you do not load the Library, you will get “unknown subprogram” errors.

Normally, you will do all of your program development with the QLB library, then make EXE files to distribute for multiple stations or other users. Remember! The Incorporated Driver License does not allow you to distribute the original library files; you can only distribute programs that incorporate the Libraries!

You will need the QB4INC.LIB file when you go to create EXE files using the QuickBASIC “Make EXE” command (Run menu), or if you want to compile and link from the DOS command line. This file should be in the current directory when you invoke the Make EXE command; QB will automatically incorporate the code into the output EXE file.

As an example, to run the supplied QBDEMO program, type:

```
      QB QBDEMO /L QB4INC    r
```

at the DOS prompt, then press f + % to run as usual. Note that no resident portion need be loaded beforehand. Be sure that the QuickBASIC header file, BASDRV.H, is present in the current directory when you load the Demo.

We have also supplied an EXE version of the QBDEMO program. To run it, type “`QBDEMO` r ” at the DOS command line. This file was made with the “Make EXE … Standalone” command on QuckBASIC’s Run menu.

Turbo PASCAL V7.0

The Turbo PASCAL binding is very similar to the standard BASIC binding. The only differences concern the actual driver call. Again, single-user and Incorporated (non-copy-protected) versions of the Driver are available.

If you are using a Single-User Linkable version of the Driver, the software will randomly check for the presence of the Hardware Key on the parallel port. If the key is not detected, the Driver will return a STATUS of -2.

The standard BASIC version documentation still applies to the Turbo binding. There are only two differences to keep in mind.

First, the Turbo binding returns the STATUS word as its return value instead of assigning it to a parameter. In other words, when you execute the **basdrv** function, it returns the STATUS value. This should be zero under normal circumstances.

Turbo PASCAL is no better than BASIC at handling variable-length parameter lists. It only permits such lists in calls to some of its built-in functions, like writeln. Therefore, **YOU MUST ALWAYS PASS A FUNCTION NUMBER AND FOUR PARAMETERS, EVEN IF NOT ALL OF THEM ARE USED**. Turbo's strict type checks will help you with this.

Second, unlike BASIC, Turbo does permit value parameters in function calls. This capability is used for the function number, which is the only parameter that is never changed by the Driver. Since you cannot mix "by reference" and "by value" variable passing in different calls, only the function number can passed by value.

A sample Turbo PASCAL program called DRVTEST.PAS is also supplied. This program permits you to monitor a table of registers on the AutoMate. To run it, load the Turbo PASCAL Driver, load Turbo PASCAL with DRVTEST as the current file, and press Alt-R to Run the sample. The program is menu-driven and should be easy to understand.

The PASDRV Unit

The Turbo 7 version of the Driver takes advantage of Turbo's "unit" capability. To make the Driver callable from a particular Pascal program, simply include the statement:

```
uses PASDRV;
```

Note that the file PASDRV.TPU, supplied on the Driver distribution disk, must be present in the current directory for this to work.

> **Note!** Turbo PASCAL units are never compatible between versions. In other words, the Version 7 unit can only be used with Turbo PASCAL Version 7. Using the wrong unit with the wrong version will cause errors or a crash.

The PASDRV "unit" contains the constant function numbers previously supplied in the DRVHDR.PAS include file. For example, the constant B_RDREG is defined in the unit as 3, the function number for Read Register. The file PASDRV.I is included so that you can see the list of constants found in the unit file.

C

The C language binding, which supports Microsoft C Version 7 and Borland C++ Version 3, is very similar to the standard BASIC interface. It is available in both Single-User Linkable and Incorporated versions. Calls to the driver take the form:

```
status = basdrv(fno,parm1,...);
```

where "fno" is an integer function number. **basdrv** returns an integer value containing an error code, if any. A return code of zero indicates that the operation was successful.

If you are using a Single-User Linkable version of the Driver, the software will randomly check for the presence of the Hardware Key on the parallel port. If the key is not detected, the Driver will return a STATUS of -2.

The function **basdrv**() should be declared as:

```
extern int _basdrv();
```

The Driver binding is found in an Object Library file called MSCDRV.LIB. This file contains the actual **basdrv()** function, and you must link it with any C program that calls the driver. MSCDRV is designed for use with the Small memory model. A second file, MSCDRVM, is provided for use with the Medium memory model. Please call ACS for instructions on interfacing with other memory models.

Note! The standard binding has no provision for operation with the Compact, Large, or Huge models. Please call us if you need to use these.

The only major difference between the C binding and the standard interface concerns value and address parameters. BASIC does not permit value parameters, so the manual says that all parameters must be passed by address.

C permits value parameters, so you may use them as often as possible. As a general rule, any scalar (int, char, etc.) that is not modified by the function should be passed by value.

Any array parameter (strings, integer arrays, etc.) must be passed by address. This is the normal convention in C. You must also pass any scalar parameter which will be altered by the Driver by address, usually by using the '&' (address of) operator.

Since the status word is returned directly by the driver, there is no need to include a "status" parameter in each call. You may discard the status word returned by **basdrv()** if it is not needed. Any nonzero status value indicates an error has occurred.

Further, BASIC and Pascal cannot handle variable-length parameter lists. When using those languages, you must always pass a fixed number of parameters to the driver, padding the list with dummies as needed. With C, you need *only pass the number of parameters required by the particular function call.*

A header file, MSCDRV.H, is included on the distribution disk to make the driver easier to use with C. This header file "`#defines`" all of the function numbers to symbols of the form "`B_name`". For example, symbol `B_WHORU` is assigned to the function number for the WHORU call. To use this file, simply put the statement “`#include <mscdrv.h>`” near the top of your C source file.

Now for some examples:

```
int idbuf[40],status;
status = basdrv(B_WHORU,idbuf);
```

This call reads the processor identification data into the integer array `idbuf`, storing the status word in the variable “`status`”.

```
int sltno;
basdrv(B_SETNOD,2);
basdrv(B_SETDSLT,sltno);
```

Sets the destination node to 2 and the destination slot to the value in `sltno`.

```
int regbuf[40],status;
status = basdrv(B_RDREG,02000,30,regbuf);
```

Reads the values of 30 registers into the integer array “`regbuf`” beginning with register 2000. The status word is saved into the integer variable “`status`”.

```
int prstat;
if (basdrv(B_RDSTAT,&prstat)) {
  puts(“Could not read processor status.”);
  return(1);
  }
printf(“Processor is %s\r\n”,(prstat) ? “RUNNING” : “Stopped”);
```

This fragment tries to read the current processor status into the variable “`prstat`”. If the operation fails, it prints an error message; otherwise, it displays the correct message.

Sample files

A sample C file, DRVCLK.C, is included to help you understand how to work with the driver. This simple program reads and displays an AutoMate processor’s clock registers on the standard output device. It also shows how to write programs that will work either directly to the AutoMate or via Gateway.

Incorporated Version

The Incorporated Driver for Microsoft C is supplied as a LIBrary file, MSCDRVI.LIB. You should include this library name in your Linker command. With the Incorporated Driver, there is no resident portion to load and no copy protection; the entire Driver is included in the library.

Visual Basic

Microsoft Visual Basic is supported by the Windows AutoMate Driver DLL. Working with the DLL is similar to using any of the other interfaces, with the following exceptions:

- **Very Important.** The Driver is available in both 16 and 32-bit versions. The 32-bit version behaves somewhat differently than the 16-bit version, which is described in this section. Please refer to the "Windows 95/NT" section on page 22 for more information on the 32-bit Driver.
- To install the DLLs, you must copy them to a directory on your system's PATH or to the Windows SYSTEM directory. You will need to explain this procedure to the end-user or use the Microsoft VB Setup application.
- The Windows AutoMate Driver DLL uses protected-mode instructions and is hence not compatible with Windows's Real mode. The DLL will only operate in Windows Standard or 386 Enhanced modes. 386 Enhanced mode provides the fastest and most reliable communications. Windows 3.1 provides markedly better serial port operation than Windows 3.0, especially at 9600 baud or higher.
- Visual Basic supports the use of Value parameters. It is not necessary to assign numeric values to variables before calling the Driver.
- The Driver is declared as a Function called **basdrv**. The Status value is returned by this function.
- Two of the Driver's functions, SRCONV and SRBCONV, require string parameters. Because Visual Basic cannot easily pass strings to DLLs, there is a "helper alias" declaration **basdrvstr** that is used when calling these two functions (see below).

Header File

ACS has supplied a set of global declarations for use with Visual Basic and the Windows Driver. These declarations are found in a text file called AMDDLL.VBH. You can install them in your Visual Basic application by opening AMDDLL.VBH with Notepad (or another text editor), selecting the entire file, copying it to the clipboard, switching to Visual Basic, and Pasting the clipboard into the VB application's Global module.

The declarations look something like this:

```
Global Const bRDPNT = 1              ' Read Value of a Point
Global Const bWRPNT = 2            ' Write Value of a Point
  and so on...
```

AMDDLL.VBH also contains the DLL declarations:

```
'
' Use these declarations while developing the application
'
```

```
Declare Function basdrv Lib “AMDDLL.DLL” (ByVal FuncNo As
    Integer, p1 As Any, p2 As Any, p3 As Any, p4 As Any) As
    Integer
Declare Function basdrvstr Lib “AMDDLL.DLL” Alias “basdrv”
    (ByVal FuncNo As Integer, ByVal p1 As String, p2 As Any, p3
    As Any, p4 As Any)
```

These declarations tell Visual Basic that the Driver functions are located in the file

Calling the DLL

For all Driver commands except SRCONV and SRBCONV, you can call the DLL with standard Visual Basic function calls like this:

```
status = basdrv(bRDREG, &O3702, 3, bar(0), 0)
```

This command reads three registers starting at address 3702 into the array **bar**. Notice that the array is passed via its first element **bar(0)**. The operation status will be returned in the variable **status**.

Note! It is *essential* that you pass arrays to the DLL by their first element. Passing an array name or **array()** will almost certainly cause a General Protection Fault (UAE under Windows 3.0).

Visual Basic requires the PASCAL calling convention to DLLs, which in essence means that you must always supply five operands to the **basdrv** function (the function number and four parameters). However, as stated above, you can pass numeric values directly and use zero (or any integer variable) as a placeholder for unused parameters.

If you need to call the SRCONV or SRBCONV commands, use a statement like this:

```
status = basdrvstr(bSRBCONV, i$, reg, bit, 0)
```

This command will convert the input string **i$** to a register and bit address. The operation status is returned in the variable **status**.

Note! **DO NOT** pass Visual Basic strings to other Driver commands, and DO NOT call SRCONV or SRBCONV with the standard **basdrv** call. Either mistake will cause a GPF or other crash.

Yielding Control

When constructing your Visual Basic application, remember to provide time for other applications to run. Windows is a *cooperative* multitasking environment; you must “voluntarily” give up control to allow other applications to execute. In the worst case, if you fail yield control, you may be unable to stop your own application.

The classic mistake of this type is to create a tight data-sampling loop like this:

```
do
```

```
	status = basdrv(bRDREG, &O2000, 60, bar(0), 0)

	for n=0 to 59
	  print#1, bar(n); ",";
	  next n

	print#1,

	loop while status = 0
```

Obviously, this loop also provides no escape other than a **status** error, but the essential point is that control never returns to Windows until this loop exits. Visual Basic will be unable to process other events for your application (or any other application).

There are two possible solutions to this problem:

- Insert a **n = DoEvents()** statement inside the **do** loop. This will allow other Windows applications to get control periodically.
- Implement this sampling function with a Visual Basic Timer control. This is the best solution. It provides very efficient operation and a controlled sampling interval. The sample application (described below) uses this approach to sample processor information. Remember that a limited number of Timer controls can be running at any given time, and that you can selectively enable or disable a Timer using its **enabled** property.

Sample Application

The Visual Basic version of the DLL is supplied with a sample Visual Basic application that demonstrates communicating with the AutoMate. The application is provided both as a standalone EXE and as a Visual Basic "project."

To run the sample application, invoke AMDDLL.EXE with the Program Manager "File / Run…" command or by double-clicking in the File Manager. The sample application will immediately try to go online and report information about the AutoMate processor.

You may also wish to examine the sample application to see how the various Driver DLL calls are used. You can do this by opening AMDDLL.MAK with the Visual Basic "File / Open Project…" command.

Windows 3.1

Windows applications other than Visual Basic can call the AutoMate Driver DLL, with the following restrictions:

- To install the DLLs, you must copy them to a directory on your system's PATH or to the Windows SYSTEM directory. You will need to explain this procedure to the end-user or provide a suitable "install" application.

- The Windows AutoMate Driver DLL uses protected-mode instructions and is hence not compatible with Windows's Real mode. The DLL will only operate in Windows Standard or 386 Enhanced modes. 386 Enhanced mode provides the fastest and most reliable communications. Windows 3.1 prvides markedly better serial port operation than Windows 3.0, especially at 9600 baud or higher.

Caution. Programming under Windows is highly demanding. Developing a Windows application is greatly more complex and unforgiving than programming under DOS. If you are not already familiar with programming under Windows, or if the information in this section seems far beyond your current knowledge, we *strongly recommend* that you consider beginning with a Windows "application generator" like Visual Basic or Delphi. Customer feedback has taught us that a simple application may take *five times or more longer* for a beginning C programmer to develop under Windows than it would under DOS.

The Driver DLL provides two caller interfaces, one that uses the `__pascal` calling convention, and one that uses the `__cdecl` convention. It is beyond the scope of this manual to describe these conventions in detail.

The `__pascal` entry is used by Visual Basic, and could be used by other applications that do not support variable-length parameter lists. Since few Driver functions use all four optional parameters, and since many Driver parameters can be efficiently passed by value, it is generally not desirable to use the `__pascal` entry point. This calling convention requires a fixed number and type of parameters; to permit differing parameter types, all parameters (except the function number) must be passed by reference.

The C declaration for this entry point is:

```
extern int __far __pascal basdrv(int fno, void __far *p1, void
   __far *p2, void __far *p3, void __far *p4);
```

Most C and C++ applications should probably use the `__cdecl` calling convention. This convention is most efficient for routines (like the Driver) that make use of variable-length parameter lists. Principally in order to provide simple compatibility for Visual Basic users, the `__cdecl` entry to the Driver is named `am_cdrv`.

The C/C++ declaration for the DLL is:

```
#ifdef __cplusplus            /* C++ definitions */
extern "C" int __far __cdecl am_cdrv(int fno, ...); /* Declare
   BASDRV */
```

```
extern “C” void __far __pascal basdrvoff(void); /* BASDRV
   Shutdown */
#else                         /* C definitions */
extern int __far __cdecl am_cdrv(int fno, ...); /* Declare
   BASDRV */
extern void __far __pascal basdrvoff(void); /* BASDRV Shutdown
   */
#endif

#define basdrv am_cdrv
#define BASDRV basdrv
```

This declaration sequence is found in the MSCDRV.H file included with the Driver. The `#define` statements allow you to use calls to `basdrv` within your application. As long as these declarations and defines are included in your program, calls to the Driver look just like calls to the DOS C-language driver described above.

Note. The Driver DLL has its own address space, as do most DLLs and DLL-located Windows API functions. It is therefore **imperative** that all pointers be sent to the Driver as `__far`. Sending `__near` pointers to the Driver will cause a General Protection Fault.

For example, a typical Driver call from a Small-model C program might look something like this:

```
int status, rval[30];
…
status = basdrv(B_RDREG, 017502, 3, (int __far *) rval);
…
```

The code fragment reads three registers into the array `rval` starting at register address 17502. The `(int __far *)` cast ensures that the location of the return-value array `rval` will be sent as a `__far` pointer.

C language support

Two support files specifically intended for use with the C language are supplied with the Windows Driver DLL. The first is the C header file MSCDRV.H. This contains the declarations for the DLL call along with `#defines` of all of the Driver function numbers. The declarations in the header file are suitable for use with Windows-compatible C and C++ compilers such as Microsoft C/C++ version 7 and Borland C/C++ Version 3.1.

The Driver DLL package also includes a DLL import library, AMDDLL.LIB. When included on the “Libraries” list sent to your compiler’s linker, this library file will provide the necessary “glue” to load and unload the DLL at runtime and prevent “symbol not found” errors during linking.

Other languages may be able to use the import library. If not, you must make the required API calls to load and unload the Driver DLL at runtime.

Note. The Driver DLL does not currently provide support for multiple callers to the DLL. It is intended for use by one calling application at a time.

Note. Currently, the Driver does not support `__huge` pointers. When calling the Driver from a Huge-model program, you will need to be sure that buffers do not cross segment boundaries.

Windows 95/NT

The AutoMate Driver is available in a 32-bit DLL for use under Windows 95 and Windows NT. Usage is very similar to the Windows 3.1 version, with the following exceptions:

- The **basdrv** function is exported from the DLL as `_stdcall`. This should make it accessible from any Visual Basic-like language. The `_cdecl` export `am_cdrv` is still available.

- The 32-bit version of the Driver is called ACD32.DLL. An import library (ACD32.LIB) is provided for users of C-like languages. The 32-bit Driver "include" file for use with C compilers is called ACD32.H.

- **Windows 95 Note:** By default, the Windows 95 Explorer is configured not to display "system" files. This means that you won't be able to "see" the Driver file ACD32.DLL in the Explorer. To make the Driver visible, go into Explorer's View menu and choose "Options…." When the Explorer Options dialog box appears, click the "Show all files" radio button in the "Hidden Files" box, then click OK.

- **<u>VITAL NOTE!</u>** All "integers" used by the Driver are *16 bits long*. The AutoMate is a 16-bit device. Visual Basic programs shouldn't have any problem with this, but C programmers must declare integers intended for the driver as `short` or `__int16`. Using `ints` *will not work* and may cause the Driver to crash.

The C declarations for the 32-bit Driver:

```
extern short _stdcall basdrv(
short fno,          // Function number
void far *p1,       // Parameter 1
void far *p2,       // Parameter 2
void far *p3,       // Parameter 3
void far *p4)       // Parameter 4

extern short am_cdrv(
short fno,          // Function number
...)                // Parameter list
```

The Visual Basic declarations for the 32-bit Driver:

```
Declare Function basdrv Lib "ACD32.dll" _
  (ByVal FuncNo As Integer, ByRef p1 As Any, _
   ByRef p2 As Any, ByRef p3 As Any, _
   ByRef p4 As Any) As Integer

Declare Function basdrvstr Lib "ACD32.dll" Alias "basdrv" _
  (ByVal FuncNo As Integer, _
   ByVal p1 As String, ByRef p2 As Any, _
   ByRef p3 As Any, ByRef p4 As Any) As Integer
```

As with the 16-bit Driver, the **basdrvstr** alias can be used with to call functions that require a string parameter.

Using the Driver over RNET

Most of the questions we receive about the Driver concern using it with the Reliance RNET local area network. It is simple enough to adapt your programs to work over RNET as long as you follow a few simple guidelines.

There are several RNET terms you should be familiar with. First, the "node number". Every device on the network, including the PC, has a node number. This number is exactly like a telephone number; it serves as a destination selector that routes your commands over the network to the appropriate destination.

Each device on the network must be assigned a different node number between 0 and 254. You should assign the node numbers in order, starting with 0. Consult your hardware manuals to determine how to set the node numbers on your equipment.

Generally, if you have only one PC on your network, it should be assigned node 0 as recommended by Reliance. The PC's node number is always the same as the Gateway (45C27) that it is connected to.

You must use the SETGWAY command (number 28) to set the Gateway's node number. When the Gateway is first powered up, it is "uninitialized" and will return Error 104 until you execute the SETGWAY command. SETGWAY should be the first command that you execute when communicating over RNET.

If you set the Gateway node to something other than 0, you must set the Origin node number to the same number. Use the SETOND command (number 52) to do this.

Once the Gateway has been initialized, you must select the destination for your commands. For the A15 and A20, only the node number is needed; use the SETNOD (number 40) to select the target processor. For A30 and A40 processors, you set both the target node (using SETNOD) and the destination slot (using SETDSLT, command number 41). For the A40, the destination slot is always the Control card, not the Logic card.

Your program may refer to as many different destinations as required. Calls will be directed to the last destination specified until you use the SETNOD and SETDSLT commands to change target processors.

RNET or Direct

It is possible to construct your program so that it will work both over RNET and through an AutoMate's front port. There are four issues to address in such programs.

1) Mode detection: is the PC connected to the Gateway or directly to the AutoMate?

2) Initializing the Gateway (if required).

3) Selecting the destination node and slot if operating over RNET.

4) Setting the destination node if the processor is directly connected.

The simplest way to handle all of these problems is to follow this procedure:

1) Set the destination node to 255 using the SETNOD command. Any device will answer a WHORU command to node 255.

2) Issue a WHORU command (number 29).

3) If BASDRV returns an Error 104 in the Status word, you are connected to an uninitialized Gateway. You must now execute the SETGWAY command before continuing. After doing so, go back to step 2. If you set the Gateway's node number to anything other than 0, you must use the SETOND command to set the PC's node to the same number.

4) Examine the first element (element 0) of the WHORU return array. This is the device's model number. The Gateway answers as Model 12.

5) If you are connected to a Model 12 device, you must select the destination node and slot and set them with the SETNOD and SETDSLT commands.

6) If you are not connected to a Model 12 device, you must set the destination node to the processor's Node Number, which is found in Element 5 of the WHORU return array. Failure to do this will cause the Driver to return a Timeout Error (-1) for any call other than WHORU.

If you follow the steps listed above, it should be relatively simple to write programs that work both in Direct mode and over RNET.

BASDRV and the Serial Comm. Card

Establishing a link with a Serial Communications Card can be a tricky business. Sadly, the card will not identify itself (via the WHORU command) until you already “know who it is.” This causes a great deal of confusion among our customers.

To communicate with an AutoMate^{tm} processor via a Serial Communications Card, you must:

1) Make sure that you are connected to a port that is in Host Computer Mode at 9600 baud. This is the default setup for Port 0 *only* on the Serial Communications Card. If you need to use one of the other ports, you must set it up first using the instructions in the card’s manual. This involves putting a function block in your AutoMate^{tm} application program that performs the setup.

2) Use the SETNOD (Set Destination Node) command to set the destination node to the ***slot*** number that the Serial Communications Card is in. This makes no sense, but it is correct.

3) Use the SETDSLT (Set Destination Slot) command to select the processor card that you want to talk to.

Once you have completed these steps, you should be able to communicate with the processor. Remember that Reliance counts slots in *octal*, but the Driver expects values in *decimal*. This makes for plenty of confusion on both the SETNOD and SETDSLT commands; you may have to try values “on either side” of the number you believe to be correct before the link will work.

Here is a sample program that links with a Serial Communications Card in Slot 3. The target processor is in Slot 1:

```
CALL BASDRV(B_SETNOD, STATUS, 3, B, B, B)    ‘ Set node to slot
CALL BASDRV(B_SETDSLT, STATUS, 1, B, B, B)   ‘ Set target slot
CALL BASDRV(B_WHORU, STATUS, BAR(0), B, B, B)     ‘ Read info
```

Error Codes

Whenever the STATUS variable is nonzero, and error has occurred. Here is a list of the codes which you may see returned in the STATUS variable:

Error Code	Meaning
-10	Serial port could not be set up
-1	Timeout (destination did not respond)
0	No error
1	Invalid function number
2	Invalid parameter
3	Address out of range
4	Illegal number of bytes requested
6	AutoMate running
7	EEPROM error (application memory)
8	Sequence not found
9	Memory protect violation
10	No memory available
11	Configuration error
12	Data register range error
61	General AutoMate hardware error*
101	Checksum error
102	Destination receive buffer full
103	Illegal command format
104	Gateway not configured
106	Slot number required

WARNING! If you have set up your serial communications card with the PC-DOS "MODE" command (for use with a serial printer, modem, etc.), the driver will not operate correctly.

* This error number may occur for many different reasons. Whenever you see Error 61, it means that the AutoMate has stopped and is in a special "error" mode. In this mode, only certain BASDRV commands are permitted; all others will return Error 61. Generally, once an Error 61 has been returned, you should only look at the system error status registers to gather information about the error. Before trying to restart the AutoMate, you should use a programming device to clear and reload its memory.

Function Quick Reference by Function Number

Function Name	Function Number	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Description
RDPNT	1	REG	BIT	DAT		Read Value of a Point
WRPNT	2	REG	BIT	DAT		Write Value of a Point
RDREG	3	STRTREG	REGCOU	INTAR		Read Value of Register(s)
WRREG	4	STRTREG	REGCOU	INTAR		Write Value of Register(s)
RDFRCT	7	RAR	BAR	FRCV	NOFRCED	Read Input Forcer Table into Register & Bit Arrays
WRFRCT	8	RAR	BAR	FRCV	NOFRCED	Write Input Forcer Table from R & B Arrays
RDSTAT	9	INTST				Read Processor Status
WRSTAT	10	INTST				Write Processor Status
RDREGLST*	11	RAR	COUNT	INTAR		Read List of Registers RAR into INTAR
WMULPT	12	COUNT	REGAR	MASKAR	DATAR	Write Multiple Points
RDREGLIM**	13	REG				Read Register Limit
WRREGLIM**	14	REG				Write Register Limit
ROFRCT**	15	RAR	BAR	FRCV	NOFRCED	Read Output Forcer Table into R & B Arrays
FRCCOIL**	16	REG	BIT	DAT		Force Coil to DAT
UNFRCCOIL**	17	REG	BIT			Unforce Coil

AutoMate Communications Driver - Function Quick Reference

INSSEQ	18	NOWRDS	INTAR			Insert Sequence at Program Pointer
DELSEQ	19					Delete Sequence at Program Pointer
SRCHSEQ	20	REG	BIT	NOWRDS	INTAR	Find Sequence
SRCHN	21	REG	BIT	NOWRDS	INTAR	Find Next Occurrence of Sequence
SRCHU	22	NOWRDS	INTAR			Get Previous Sequence
SRCHD	23	NOWRDS	INTAR			Get next sequence
SRCHTOP	24	NOWRDS	INTAR			Get First Sequence
CHKSEQ	25	REG	BIT	NOWRDS	INTAR	Check For a Sequence
CHKN	26	REG	BIT	NOWRDS	INTAR	Check for next occurrence of Sequence
SETRNET	27	NODES				Set Number of Nodes on RNET
SETGWAY	28	NODENO	NODES	CONFIG		Set Gateway Parameters
WHORU	29	INTAR				Read Identifying Information
SETCOMM	30	NODENO	NODES			Set Communication Parameters
CLRMEM	31					Clear Application Memory
MEMUSE	32	LONGAR				Read Memory Use Statistics
RDIOCFG**	33	INTAR				Read I/O Configuration Table
WRIOCFG**	34	INTAR				Write I/O Configuration Table
REQACC*	35					Request Protected Access

AutoMate Communications Driver - Function Quick Reference

CANACC*	36				Cancel Protected Access
MEMDIAG	37	INTAR			Run Memory Diagnostic
IODIAG	38	INTAR			Run I/O Diagnostic
GWAYDIAG	39	NOWRDS	INTAR	OUTAR	Run Gateway Diagnostic
SETNOD	40	NODE			Set Destination Node
SETDSLT	41	SLOT			Set Destination Slot
SETSSLT	42	SLOT			Set Origin Slot Number
SETBAUD	43	RATE			Set Baud Rate
SETSNG	44	FLAG			Set Single-Processor Mode
CLROFRC**	45				Clear Output Forcer Table
AUTOCOM	46	INTAR			Establish Communications
SRBCONV	47	STR	REG	BIT	Convert string to register and bit
SRCONV	48	STR	REG		Convert string to register
SETDLA	49	COUNT			Set Communications Delay
WORDAR	50	DAT	INTAR		Unpack Integer to Array
ARWORD	51	INTAR	DAT		Pack Array into Word
SETOND	52	NODE			Set origin (PC) node
SETMSK	53	MASK			Set Interrupt Mask

AutoMate Communications Driver - Function Quick Reference

STPORT	54	PORTN				Select Comm. Port
RDCHEK*	55	APPC	K20C	IOC		Read AutoMate Checksums
STEXT	56	ITEXT				Send an ASCII string to the serial port
FMO2MS	57	COUNT	MOTFLAR	MSFLAR		Convert Motorola floating point to Microsoft
FMS2MO	58	COUNT	MSFLAR	MOTFLAR		Convert Microsoft floating point to Motorola
FRMPRO=	59	FLAG				Set Multitasking Frame
OFF	60					Deactivate Driver
PSYST=	61	BASEADR	XMODE	RMODE	CHAINF	Serial Port System Setup
KEYPORT	63	PORTN				Select Hardware Key port (LPT1 to LPT3)
PCLINK	64	BASEADR	NODENO	NODES		Set up Reliance R-Net PC Link Card

Legend:

*	Not valid for A15
**	Not valid for A15 or A20
=	Not valid for Windows DLLs

Variable Names used in Quick Reference Table

Identifier	**Type**	**Description**
APPC	Integer	Application Program Checksum
BAR	Integer Array	Array of Bit Numbers
BIT	Integer	Bit Number
CONFIG	Integer	Gateway CONFIG byte (See Gateway Manual)
COUNT	Integer	Count
DAT	Integer	Bit Value
DATAR	Integer Array	Array of Bit Values
FLAG	Integer	Mode Selector (1 or 0)
FRCV	Integer Array	Array of point force values
INTAR	Integer Array	Array of data words or bytes
INTST	Integer	Processor status word
IOC	Integer	I/O Configuration Checksum
ITEXT	String	ASCII string to send to serial port
K20C	Integer	20000 register checksum
LONGAR	Long Integer Array	Array of Longwords (values may exceed 32767)
MASK	Integer	Interrupt mask word
MASKAR	Integer Array	Array of bit masks
MOTFLAR	Floating Point Array	Array of Motorola floating point values
MSFLAR	Floating Point Array	Array of Microsoft floating point values
NODE	Integer	Node Number (0 to 254)
NODENO	Integer	Node Number (0 to 254)
NODES	Integer	Number of nodes (1 to 254)

NOFRCED	Integer	Number of entries in forcer table
NOWRDS	Integer	Number of words to transmit
OUTAR	Integer Array	Gateway Diagnostic return array
PORTN	Integer	Communications port number (1 or 2)
RAR	Integer Array	Register number array
RATE	Integer	Baud Rate code (0 to 7)
REG	Integer	Register Number
REGAR	Integer Array	Array of register numbers
REGCOU	Integer	Number of registers to move
SLOT	Integer	Slot Number (0 to 20)
STR	String	String to decode
STRTREG	Integer	Starting register number

Function Summary

The rest of this manual consists of language interface descriptions and synopses of BASDRV's functions. The variable B is used as a dummy placeholder to ensure that six parameters are always passed to BASDRV from BASIC, which has no variable-length argument list provisions.

This section contains a "synopsis" of each function supported by the AutoMate driver. Note also that the function synopses are written for the Interpreted BASIC / Microsoft QuickBASIC version of the Driver. The other language interfaces use exactly the same function numbers and parameters; the only difference between language interfaces is the "form" of the function calls.

The AutoMate driver uses a total of four different variable types: Integer, Integer Array, Unsigned Integer, Long Integer Array, and String. Integers are always 16-bit words. Strings are arrays of bytes less than 255 characters long. You will simply use the Integer, Unsigned Integer, and String types built into your language; the language interface will take care of any necessary translation.

Remember! If a synopsis lists less than 4 ARGuments (in addition to the STATUS and FUNCTION NUMBER parameters) you may still have to pass all four if you are using a language like BASIC or Turbo PASCAL that does not permit variable-length parameter lists. For these languages, you must "pad" the argument list out to four ARGuments with "dummy" variables ("B" in the synopses). Please consult the Language Interfaces section for more information on this topic.

Since these synopses are written for the "lowest common denominator," Interpreted BASIC, they always include the full 6-parameter list and always assign values to variables, then pass the variables. Some languages, such as C and QuickBASIC, do not require all of these steps. Check the binding description for your language to see exactly what you must do to make a call to the Driver.

RDPNT --- Read Value of a Point

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 1	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register	Integer
ARG2	Bit	Integer
ARG3	Value of point	Integer
ARG4		

DESCRIPTION

Returns the value of the point described by ARG1 and ARG2 in ARG3.

EXAMPLE

```
REG = &O2000
BIT = &O14
DAT = 0
CALL BASDRV(B.RDPNT, STATUS, REG, BIT, DAT, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"State of point 2000.14 is"DAT
```

WRPNT --- Write Value to a Point

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 2	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register Number	Integer
ARG2	Bit Number	Integer
ARG3	New value for point	Integer
ARG4		

DESCRIPTION

Sets point described by ARG1 and ARG2 to the value in ARG3.

EXAMPLE

```
REG = &O2000
BIT = &O14
DAT = 1
CALL BASDRV(B.WRPNT, STATUS, REG, BIT, DAT, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Point 2000.14 set to 1"
```

RDREG --- Read Register Values

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 3	Function Number	Integer
STATUS	Return Code	Integer
ARG1	First register to read	Integer
ARG2	No. of regs. to read	Integer
ARG3	Dest. for value(s)	Integer array
ARG4		

DESCRIPTION

Reads ARG2 consecutive registers beginning with the register number contained in ARG1. The register values are stored in consecutive elements of the integer array ARG3. If only one register is being read, ARG3 can be an ordinary integer.

COMMENTS

ARG2 must be less than 122.

EXAMPLE

```
REG = &O2000
COU = 10
DIM RAR(10)
CALL BASDRV(B.RDREG, STATUS, REG, COU, RAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Registers 2000-2011:"
FOR LA=0 TO 9:PRINT OCT$(&O2000+LA),RAR(LA)):NEXT
```

WRREG --- Write Values to Registers

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 4	Function Number	Integer
STATUS	Return Code	Integer
ARG1	First reg. to write	Integer
ARG2	No. of regs. to write	Integer
ARG3	Array of values	Integer Array
ARG4		

DESCRIPTION

Writes values from consecutive elements of integer array ARG3 to ARG2 consecutive registers beginning with register ARG1.

COMMENTS

Writes to up to 122 registers.

EXAMPLE

```
REG = &O2000
COU = 10
DIM RAR(10)
FOR LA=0 TO 9:RAR(LA)=56:NEXT
CALL BASDRV(B.WRREG, STATUS, REG, COU, RAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Registers 2000-2011 set to 56"
```

RDFRCT --- Read Input Forcer Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 7	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register array	Integer array
ARG2	Bit array	Integer array
ARG3	Force values	Integer array
ARG4	No. of points forced	Integer

DESCRIPTION

Reads the contents of the processor’s input force table. After the read, arrays ARG1 and ARG2 describe the points that are forced. Array ARG3 contains the force values, and ARG4 indicates how many points are forced.

COMMENTS

ARG4 can vary from 0 to 20.

EXAMPLE

```
DIM RAR(20),BAR(20),FRCV(20)
COU = 1
CALL BASDRV(B.RDFRCT, STATUS, RAR(0), BAR(0), FRCV(0), COU)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
COU = COU-1
PRINT"Input Forcer Table"
FOR LA=0 TO COU
  PRINT OCT$(RAR(LA))"."OCT$(BAR(LA))" Forced To"FRCV(LA)
NEXT LA
```

WRFRCT --- Write Input Forcer Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 8	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Registers	Integer array
ARG2	Bits	Integer array
ARG3	Force values	Integer array
ARG4	No. of points to force	Integer

DESCRIPTION

Loads the processor’s input forcer table. To use, load the integer arrays ARG1 and ARG2 with the point numbers to force and load integer array ARG3 with the desired force values. ARG3 should contain only values of zero or one, although any nonzero value will be interpreted as a one. Set ARG4 to the number of points, and execute the call.

Please refer to page for information and restrictions on forcing.

COMMENTS

Note that this function clears and rewrites the Input Forcer Table on each call. If you wish to force or unforce individual points, you must read in the table, add or delete the desired point, and rewrite the table.

ARG4 must be between 0 and 20. A value of zero clears the table.

EXAMPLE

```
REG = &O14
BIT = &O12
COU = 1
DAT = 1
CALL BASDRV(B.WRFRCT, STATUS, REG, BIT, DAT, COU)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Input 14.12 is now the only input forced.  It"
PRINT"is forced to 1."
```

Note: You can use integers in place of integer arrays if only one point is being forced or if you wish to clear the forcer table by passing COU=0.

RDSTAT --- Read Processor Status

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 9	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Processor Status	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Reads the current status of the AutoMate processor into ARG1. Two values are possible: 1 indicates Processor Running, 0 means Processor Halted.

EXAMPLE

```
DAT = 0
CALL BASDRV(B.RDSTAT, STATUS, DAT, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
IF DAT=1 THEN PRINT"Processor Running" ELSE PRINT"Processor
    Halted"
```

WRSTAT --- Write Processor Status

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 10	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New status	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the processor status. ARG1 can be set to:

0	Stop Processor
1	Run Processor
2	Single Scan

before the call.

COMMENTS

Note! This command changes the processor status immediately, without any further confirmation. Be sure that you actually wish to alter the status before executing this command!

The processor will execute one scan for each time that you call WRSTAT with ARG1 = 2.

EXAMPLE

```
PRINT"*** Run Processor ***"
PRINT"Are you sure ?"
IF INPUT$(1)="Y" THEN
  NEWSTA = 1:
  CALL BASDRV(B.WRSTAT, STATUS, NEWSTA, B, B, B):
  IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

RDREGLST --- Read List of Registers

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 11	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Registers to read	Integer array
ARG2	No. of regs. to read	Integer
ARG3	Register values	Integer array
ARG4		

DESCRIPTION

This function allows you to read multiple nonconsecutive registers. To use it, place the numbers of the registers in consecutive elements of the integer array ARG1 and the count in ARG2. After the call, the array ARG3 will contain the values.

COMMENTS

Not allowed on A15. You may specify up to 122 register numbers to read.

EXAMPLE

```
RESTORE
COU = 0
DIM RAR(122),VAR(122)
READ RAR(0)
WHILE RAR(COU)>0
  COU = COU+1
  READ RAR(COU)
  WEND
CALL BASDRV(B.RDREGLST, STATUS, RAR(0), COU, VAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Register","Value"
COU=COU-1
FOR LA=0 TO COU
  PRINT OCT$(RAR(LA)),VAR(LA)
  NEXT LA
DATA &O10,&O40,&O2000,&O20000,-1
```

WMULPT --- Write Multiple Points

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 12	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Count	Integer
ARG2	Register List	Integer array
ARG3	Mask List	Integer array
ARG4	Data	Integer array

DESCRIPTION

This function provides a method to modify multiple points without repeated calls to WRPNT. It uses binary masks to determine which points in a given register will be affected. For example, suppose that:

```
ARG2(0) = &O2000  [ Selects Register 2000 ]
Reg 2000 (Before) = &HACAC
ARG3(0) = &HFF    [ Only lower byte will be affected ]
ARG4(0) = &H6565  [ Mask ]
```

The results will be:

```
Reg. 2000 = 1010 1010  1010 1010
Mask      = 0000 0000  1111 1111
Data      = 0101 0101  0101 0101
________________________________
Result    = 1010 1010  0101 0101
```

The new result, &HAC65, will be assigned to register 2000.

COMMENTS

Up to 40 registers can be written in one call.

EXAMPLE

To execute the above example assuming Register 2000 already contains &HACAC:

```
REG = &O2000
MASK = &HFF
DAT = &H6565
COU = 1
```

```
CALL BASDRV(B.WMULPNT, STATUS, COU, REG, MASK, DAT)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

Note: We have once again used integers in place of integer arrays as only one register is being modified.

RDREGLIM --- Read Register Limit

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 13	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Current Register Limit	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

A20E, A30 and A40 only. Reads the current data register limit and returns it in ARG1. The data register limit is always 20000 Octal or higher.

EXAMPLE

```
REG = 0
CALL BASDRV(B.RDREGLIM, STATUS, REG, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Current Register Limit is "OCT$(REG)
```

WRREGLIM --- Write Register Limit

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 14	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New Register Limit	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

A20E, A30 and A40 only. Tries to set the new user data area register limit to ARG1. Will fail if the AutoMate processor does not have enough free memory to create the new number of 20000 data registers.

You can use the MEMUSE command to determine if there is enough free memory to create new registers.

EXAMPLE

```
REG = &O20100
CALL BASDRV(B.WRREGLIM, STATUS, REG, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"New register limit set"
```

ROFRCT --- Read Output Forcer Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 15	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Registers	Integer array
ARG2	Bits	Integer array
ARG3	Force Values	Integer array
ARG4	No. of points forced	Integer

DESCRIPTION

A30 or A40 only. Reads the current contents of the processor's Output Forcer Table into arrays ARG1, ARG2, and ARG3. After the call, ARG4 will contain the number of points in the table.

Forcing is a complicated operation. This command may or may not show all outputs that are forced. The details of the output forcing procedure are beyond the scope of this manual.

COMMENTS

The table can contain up to 20 points.

EXAMPLE

```
COU = 0
DIM RAR(20),BAR(20),FRCV(20)
CALL BASDRV(B.ROFRCT, STATUS, RAR(0), BAR(0), FRCV(0), COU)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
COU = COU-1
PRINT"Output Forcer Table:"
FOR LA=0 TO COU
  PRINT OCT$(RAR(LA))"."OCT$(BAR(LA))" Forced To"FRCV(LA)
NEXT LA
```

FRCCOIL --- Force Output

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 16	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register number	Integer
ARG2	Bit number	Integer
ARG3	Force value	Integer
ARG4		

DESCRIPTION

A30 or A40 only. Forces the output point described by ARG1 and ARG2 to the value in ARG3.

Forcing is a complicated operation. This command will only properly force digital outputs that do not have coils in the ladder program. Forcing outputs that do have coils in the application program requires modifying the ladder rung in question and is beyond the scope of this manual.

EXAMPLE

```
REG = &O10
BIT = &016
DAT = 1
CALL BASDRV(B.FRCCOIL, STATUS, REG, BIT, DAT, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Output 10.16 Forced On"
```

UNFRCCOIL --- Unforce Output

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 17	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register	Integer
ARG2	Bit	Integer
ARG3		
ARG4		

DESCRIPTION

A30 or A40 only. Unforces the output point described by ARG1 and ARG2. See page 48 for comments on forcing.

EXAMPLE

```
REG = &O10
BIT = &O16
CALL BASDRV(B.UNFRCCOIL, STATUS, REG, BIT, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Coil 10.16 is no longer forced"
```

INSSEQ --- Insert Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 18	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Number of words	Integer
ARG2	Sequence	Integer array
ARG3		
ARG4		

DESCRIPTION

Inserts the ladder sequence (of length ARG1) described by the words in the integer array ARG2 at the current position of the program pointer. If there are already sequences at this location, they will be moved down to make room.

COMMENTS

The ladder sequence must be in internal form for the insertion to be successful. Advanced knowledge beyond the scope of this manual is required to encode and decode ladders in the required format. Use this command (and all others that modify the program memory) with great caution.

EXAMPLE

Assuming that a correctly encoded ladder of length 6 words is found in the integer array LADAR (which was dimensioned to size 50):

```
NOWRDS = 6
CALL BASDRV(B.INSSEQ, STATUS, NOWRDS, LADAR(0), B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"New Ladder Inserted"
```

DELSEQ --- Delete Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 19	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Delete ladder sequence at the program pointer.

COMMENTS

There is no way to recover the sequence deleted by this command. Use this command with great caution.

EXAMPLE

```
CALL BASDRV(B.DELSEQ, STATUS, B, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Sequence deleted."
```

SRCHSEQ --- Find Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 20	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register number	Integer
ARG2	Bit number	Integer
ARG3	No. of words	Integer
ARG4	Storage for ladder	Integer array

DESCRIPTION

Searches the application memory for the sequence described by ARG1 and ARG2 beginning with the first sequence in the program. If the sequence exists, BASDRV will return it in the integer array ARG4. The sequence's length will be returned in ARG3. The processor's current position pointer is moved to point to the sequence.

COMMENTS

The integer array ARG4 must be dimensioned (122) or more to accommodate the largest possible ladder sequence.

If the sequence is not found, STATUS will be equal to 8. If the sequence does exist, it will be returned in the array ARG4 in internal format, one word in each consecutive element. Decoding ladder sequences is beyond the scope of this manual. For more information, contact Reliance Electric directly.

EXAMPLE

```
REG = &O2000 : BIT = &O13 : NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.SRCHSEQ, STATUS, REG, BIT, NOWRDS, INTAR(0))
IF STATUS=8 THEN PRINT"Sequence 2000.13 not found."
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Sequence 2000.13 found. Contents:"
NOWRDS = NOWRDS-1
FOR LA=0 TO NOWRDS
  PRINT " "RIGHT$("000"+HEX$(INTAR(LA)),4);
NEXT LA
```

SRCHN --- Find Next Occurrence of Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 21	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register	Integer
ARG2	Bit	Integer
ARG3	No. of words	Integer
ARG4	Storage for Sequence	Integer array

DESCRIPTION

Searches for the sequence described by ARG1 and ARG2 beginning at the current program pointer instead of the first sequence in memory. If the sequence is found, the processor's program pointer is moved to point to it.

COMMENTS

Same as SRCHSEQ, page 52. Use this function to retrieve coil numbers that occur more than once, such as LOOP / END pairs.

EXAMPLE

```
REG = &O2000
BIT = &O13
DIM INTAR(122)
NOWRDS = 0
CALL BASDRV(B.SRCHN, STATUS, REG, BIT, NOWRDS, INTAR(0))
IF STATUS=8 THEN PRINT"Second occurrence not found."
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Second occurrence of sequence 2000.13 found."
```

SRCHU --- Get Previous Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 22	Function Number	Integer
STATUS	Return Code	Integer
ARG1	No. of words	Integer
ARG2	Storage for sequence	Integer array
ARG3		
ARG4		

DESCRIPTION

Gets the sequence immediately before the current program pointer position into ARG2. The program pointer is also moved back to point to the previous sequence.

COMMENTS

Same as SRCHSEQ, page 52. Returns STATUS=8 if the program pointer is at the first sequence or if the processor's application memory is empty.

EXAMPLE

```
NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.SRCHU, STATUS, NOWRDS, INTAR(0), B, B)
IF STATUS=8 THEN PRINT"No previous sequence"
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Previous sequence located."
```

SRCHD --- Get Next Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 23	Function Number	Integer
STATUS	Return Code	Integer
ARG1	No. of words	Integer
ARG2	Storage for sequence	Integer array
ARG3		
ARG4		

DESCRIPTION

Moves the processor's program pointer down one sequence and returns that sequence.

COMMENTS

Same as SRCHSEQ, page 52. Returns STATUS=8 if the pointer is at the end of the program or if the application memory is empty.

EXAMPLE

```
NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.SRCHD, STATUS, NOWRDS, INTAR(0), B, B)
IF STATUS=8 THEN PRINT"End of program."
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Next statement located"
```

SRCHTOP --- Get First Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 24	Function Number	Integer
STATUS	Return Code	Integer
ARG1	No. of words	Integer
ARG2	Storage for sequence	Integer array
ARG3		
ARG4		

DESCRIPTION

Moves the processor’s program pointer to the beginning of application memory and returns the first sequence in the program.

COMMENTS

Same as SRCHSEQ, page 52. Returns STATUS=8 if the application memory is empty.

EXAMPLE

```
NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.SRCHTOP, STATUS, NOWRDS, INTAR(0), B, B)
IF STATUS=8 THEN PRINT"Memory empty."
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"At top of program."
```

CHKSEQ --- Check for a Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 25	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register number	Integer
ARG2	Bit number	Integer
ARG3	No. of words	Integer
ARG4	Storage for sequence	Integer array

DESCRIPTION

This function is the same as SRCHSEQ except the processor's program pointer is not moved. The search always begins with the first sequence in memory.

COMMENTS

Same as SRCHSEQ, page 52. Returns STATUS=8 if the sequence does not exist.

EXAMPLE

```
REG = &O2000
BIT = &013
NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.CHKSEQ, STATUS, REG, BIT, NOWRDS, INTAR(0))
IF STATUS=8 THEN PRINT"Sequence not found"
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Sequence exists."
```

CHKN --- Check for next Occurrence of Sequence

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 26	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Register number	Integer
ARG2	Bit number	Integer
ARG3	No. of words	Integer
ARG4	Storage for sequence	Integer array

DESCRIPTION

This function is the same as SRCHN except the processor's program pointer is not moved. The search for the sequence described by ARG1 and ARG2 begins at the current program pointer position. If an occurrence of the sequence can be found between the program pointer and the end of the program, it will be returned in ARG4.

COMMENTS

Same as SRCHN, page 53. Returns STATUS=8 if the sequence (or another occurrence of the sequence) does not exist.

EXAMPLE

```
REG = &O2000
BIT = &O13
NOWRDS = 0
DIM INTAR(122)
CALL BASDRV(B.CHKN, STATUS, REG, BIT, NOWRDS, INTAR(0))
IF STATUS=8 THEN PRINT"No more occurrences found."
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Another occurrence found."
```

SETRNET --- Set Number of Nodes on RNET

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 27	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Number of nodes	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the number of nodes on an RNET Local Area Network.

COMMENTS

ARG1 should be set to one more than the highest node number on the network.

EXAMPLE

```
NODES = 2
CALL BASDRV(B.SETRNET, STATUS, NODES, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Nodes 0 and 1 enabled."
```

SETGWAY --- Set Gateway Parameters

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 28	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Node number	Integer
ARG2	Number of nodes	Integer
ARG3	Config byte	Integer
ARG4		

DESCRIPTION

Sets up an RNET Gateway unit. ARG1 determines the node number of the Gateway itself, and ARG2 sets the highest node number on the network.

ARG3 sets the Gateway's Config byte. This byte can be used to lengthen the Gateway's timeout delay if you frequently experience "Timeout Errors" over RNET. For information on this byte, you should consult your Gateway manual or contact Reliance.

COMMENTS

ARG2 should be set to the highest node number on the network plus one.

If you receive an error 104 (Gateway not configured) when trying to communicate with a processor via a Gateway, you should issue this command.

EXAMPLE

```
NODENO = 0
NODES = 2:REM Highest Node + 1 is 2
CONFIG = 0:REM No delay extension
CALL BASDRV(B.SETGWAY, STATUS, NODENO, NODES, CONFIG, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Gateway set up as node 0. Node 1 enabled."
```

WHORU --- Read Identifying Information

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 29	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Storage for data	Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

This command allows you to identify the processor type that the computer is connected to, its node number, as various other parameters. These parameters are returned in the first 14 elements of the integer array ARG1 with one parameter in each element.

COMMENTS

After a successful return, the array ARG1 will contain the following 14 parameters:

Element	Description
0	Model Number:
	9 = AutoMax
	12 = Gateway
	15 = AutoMate 15
	18 = AutoMate 15E
	20 = AutoMate 20
	21 = AutoMate 20E
	29 = AutoMate 30
	30 = AutoMate 30E
	38 = AutoMate 40X
	39 = AutoMate 40
	40 = AutoMate 40E
1	Memory Protect Status:
	bit 2 = 0: Privileged Access

	bit 2 = 1: No Access
2	Processor Software Version (X.x)
3	Processor Software Revision (x.X)
4	Processor Software Release
5	Processor Node Number
6	Number of Nodes Set
7	Not Used
8	Number of RNET CRC Errors
9	Number of RNET Overrun Errors
10	Internal Use Only
11	Internal Use Only
12	Internal Use Only
13	RNET Token Status:
	bit 0 Self Test in Progress
	bit 1 Initialize
	bit 2 Watching Bus
	bit 5 Using Token
	bit 6 Passing Token
	bit 11 Loopback Test Failure
	bit 12 Solid-state Switch Failure

EXAMPLE

```
DIM INTAR(14)
CALL BASDRV(B.WHORU, STATUS, INTAR, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Processor model"INTAR(0)"node"INTAR(5)
```

SETCOMM --- Set Communication Parameters

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 30	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Node number	Integer
ARG2	Number of nodes	Integer
ARG3		
ARG4		

DESCRIPTION

Use this function to set the processor’s internal node number and the number of nodes that it should recognize. This command affects only communications through the front (programming) port.

COMMENTS

ARG2 should be set to the highest node number on the network plus one.

EXAMPLE

```
NODENO = 1
NODES = 2
CALL BASDRV(B.SETCOMM, STATUS, NODENO, NODES, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Processor set to node 1 of 3"
```

CLRMEM --- Clear Application Memory

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 31	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Clears the application memory and forcer tables. The I/O configuration is not modified.

COMMENTS

WARNING! THIS FUNCTION WIPES THE ENTIRE APPLICATION MEMORY. THERE IS NO WAY TO RECOVER THE CONTENTS OF THE PROCESSOR'S MEMORY IF YOU DO THIS.

EXAMPLE

```
CALL BASDRV(B.CLRMEM, STATUS, B, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Memory cleared."
```

MEMUSE --- Read Memory Use Statistics

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 32	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Returned data	Long Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

This function returns four values describing the processor's memory status in the first four elements of the integer array ARG1.

Note that the values returned are long integers (32 bits). Under Interpreted BASIC, you must convert to floating point and scale pairs of array elements to get the correct values; for other languages, declare the output array as long integer.

It is important to remember that the values are returned by MEMUSE are in bytes but the AutoMate^{tm} consumes memory in words, with each word consisting of two bytes.

COMMENTS

The four values returned are:

Element	Description
0	Memory Size
1	EEPROM bytes used
2	R/W bytes used
3	Number of Power Failures

The Memory Size value is only of interest on processors with more than one possible memory configuration. To interpret the Memory Size value, use the following table:

Processor	Memory Size	Total Memory (EEPROM - Read/Write)
A15	-	2K - 4K
A20	-	4K - 12K
A30	0	4K - 4K
	1	8K - 8K
A30E	0	8K - 8K
	1	16K - 16K
A40	1	16K - 16K
A40E	n	n * 16K - n * 16K

EXAMPLE

```
DIM INTAR(4)
CALL BASDRV(B.MEMUSE, STATUS, INTAR(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT 1.0*INTAR(1)+65536.0*INTAR(2)"bytes of application memory
  free."
```

RDIOCFG --- Read I/O Configuration Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 33	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Storage for Table	Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

A30 or A40 only. Reads the processor's I/O Configuration Table into consecutive elements of the integer array ARG1.

COMMENTS

The array ARG1 should be dimensioned to at least 902 bytes long in order to accommodate the largest possible A40 configuration table. The A30 configuration table is 150 bytes long.

For information on how to decode the I/O Configuration Table, consult the Reliance Electric AutoMate Communications Protocol Manual.

EXAMPLE

```
DIM INTAR(75)
CALL BASDRV(B.RDIOCFG, STATUS, INTAR(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Configuration table:"
FOR LA=0 TO 74
  PRINT " "RIGHT$("0000"+HEX$(INTAR(LA)),4)
  NEXT LA
```

WRIOCFG --- Write I/O Configuration Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 34	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New configuration	Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

A30 or A40 only. Writes the contents of the integer array ARG1 to the processor's I/O Configuration table.

COMMENTS

This command always completely overwrites the current contents of the configuration table. The length written is determined by the data in the table.

For information on how to decode the I/O Configuration Table, consult the Reliance Electric AutoMate Communications Protocol Manual.

Warning! An incorrectly formatted configuration table can cause the processor to crash. Use this command with great caution.

EXAMPLE

Assuming that a valid configuration is already found in the integer array NEWCONF:

```
CALL BASDRV(B.WRIOCFG, STATUS, NEWCONF(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Configuration table written."
```

REQACC --- Request Protected Access

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 35	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Requests exclusive access to an AutoMate processor.

COMMENTS

You must execute this function before you can alter any processor memory locations, either registers or application memory.

EXAMPLE

```
CALL BASDRV(B.REQACC, STATUS, B, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Access OK."
```

CANACC --- Cancel Protected Access

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 36	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Releases exclusive access to an AutoMate processor.

COMMENTS

We recommend that you execute this command when you are done communicating with a processor in order that other devices may gain access to it. You may wish to do this as part of the exit procedure from your program.

EXAMPLE

```
CALL BASDRV(B.CANACC, STATUS, B, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Protected Access Released."
END
```

MEMDIAG --- Run Memory Diagnostic

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 37	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Return array	Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

Runs a nondestructive test on the AutoMate processor’s memory. This function returns four codes in the first four elements of the array ARG1.

COMMENTS

The processor must be halted before you can run this test.

The four return values are for the Scratchpad, R/W, EEPROM, and NVRAM memories respectively. A nonzero return value indicates that the corresponding memory area failed its test.

EXAMPLE

```
DIM INTAR(4),OK$(2)
CALL BASDRV(B.MEMDIAG, STATUS, INTAR(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
OK$(0) = "Passed."
OK$(1) = "Failed."
PRINT"Scratchpad memory "OK$(SGN(INTAR(0))
PRINT"R/W        memory "OK$(SGN(INTAR(1))
PRINT"EEPROM     memory "OK$(SGN(INTAR(2))
PRINT"NVRAM      memory "OK$(SGN(INTAR(3))
```

IODIAG --- Run I/O Diagnostic

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 38	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Return array	Integer array
ARG2		
ARG3		
ARG4		

DESCRIPTION

This function tests the communications between the AutoMate and the I/O Rail or Local Head. It returns a result code in each of the first four elements of the integer array ARG1, one code per I/O port.

COMMENTS

A15 Only! The A15 processor must be halted before this test can be run.

Each result code can take on the following values:

Value	Description
0	Port and Device passed test
1	I/O Reset Error
2	Clock Error
3	Serial Data Out Error
255	No rail connected

EXAMPLE

```
DIM INTAR(4)
CALL BASDRV(B.IODIAG, STATUS, INTAR(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
FOR LA=0 TO 3
    IF INTAR(LA)<>0 THEN PRINT"Port"LA"failed test."
NEXT LA
```

GWAYDIAG --- Run Gateway Diagnostic

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 39	Function Number	Integer
STATUS	Return Code	Integer
ARG1	No. of words to send	Integer
ARG2	Send array	Integer array
ARG3	Return array	Integer array
ARG4		

DESCRIPTION

Runs the Gateway loopback test. The ARG1 data words in the send array ARG2 are sent to the Gateway and echoed back into the return array ARG3. If the test was successful, arrays ARG2 and ARG3 should be identical after the call.

COMMENTS

ARG1 cannot exceed 38.

EXAMPLE

```
DIM SAR(30),RAR(30)
COU = 30
FOR LA=1 TO COU
  SAR(LA)=RND*32767
  NEXT LA
CALL BASDRV(B.GWAYDIAG, STATUS, COU, SAR(1), RAR(1), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
FOR LA=1 TO COU
  IF SAR(LA)<>RAR(LA) THEN PRINT"Test failed.":STOP
  NEXT LA
```

SETNOD --- Set Destination Node

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 40	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Node number	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the node to which AutoMate command strings are sent. Also used with the Serial Communications Card.

COMMENTS

If the computer is properly connected to the AutoMate and you are receiving "Timeout" errors, the destination node or destination slot number is probably wrong.

EXAMPLE

```
NODE = 1
CALL BASDRV(B.SETNOD, STATUS, NODE, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Destination node set to 1."
```

SETDSLT --- Set Destination Slot

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 41	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Dest. Slot No.	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the AutoMate chassis slot number to which command strings are sent.

COMMENTS

The default destination slot is Slot 1.

EXAMPLE

```
SLOT = 2
CALL BASDRV(B.SETDSLT, STATUS, SLOT, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Communicating with processor in Slot 2"
```

SETSSLT --- Set Originating Slot Number

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 42	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New Org. Slot no.	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the slot number from which command strings are originating.

COMMENTS

This should be set to the slot number of whatever device the computer is connected to.

Normally, there is no need to use this call as it is ignored by the processor.

EXAMPLE

```
REG = &O2000
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

SETBAUD --- Set Communication Rate

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 43	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Rate Code	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the transmission rate used to communicate with the AutoMate processor.

COMMENTS

The following values are allowed for ARG1:

Value	Baud Rate
0	110
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	9600
8	19,200

The default value is 7 (9600 baud).

EXAMPLE

```
BAUD = 5
CALL BASDRV(B.SETBAUD, STATUS, BAUD, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Baud Rate set to 2400".
```

SETSNG --- Set Single-Processor Mode

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 44	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Mode flag	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

If ARG1 is nonzero, BASDRV will use short (single processor) frames to communicate with the AutoMate. If ARG1 is zero, the driver will send long (multiple processor) frames.

COMMENTS

Short frames are normally only used with older versions of the AutoMate 15. All current AutoMate^{tm} processors support long frames.

EXAMPLE

```
A15FLAG = 1
CALL BASDRV(B.SETSNG, STATUS, A15FLAG, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
IF A15FLAG THEN PRINT"Sending short frames"
```

CLROFRC --- Clear Output Forcer Table

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 45	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Clears the processor's Output Forcer Table. This may or may not actually clear all outputs forced; please see the description of FRCCOIL, page 48.

EXAMPLE

```
CALL BASDRV(B.CLFOFRC, STATUS, B, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"No outputs forced."
```

AUTOCOM --- Establish Communications

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 46	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Return array	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Establishes communication with an AutoMate device through the programming port. This command will automatically set the frame length and destination node if possible. After the call, the integer array ARG1 will contain the same data as that returned by WHORU.

COMMENTS

If you are connecting directly to the AutoMate processor's front port, we recommend this command as the simplest way to establish communications. If you execute AUTOCOM before any other calls, you should not have to worry about any communications parameters (like destination node, etc.).

You must use the manual setup (SETNOD, SETDSLT, etc.) instructions if you are communicating over R-Net or via a Serial Communications Card.

EXAMPLE

```
DIM INTAR(14)
CALL BASDRV(B.AUTOCOM, STATUS, INTAR(0), B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Communications Established."
PRINT"Processor model number"INTAR(0)
```

SRBCONV --- Convert String to Register and Bit

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 47	Function Number	Integer
STATUS	Return Code	Integer
ARG1	String to convert	String
ARG2	Register return	Integer
ARG3	Bit return	Integer
ARG4		

DESCRIPTION

Since BASIC provides no capability to convert an octal string to binary, ACS has provided two functions to handle conversion and error checking on octal input strings.

COMMENTS

SRBCONV converts a string in the form "register.bit" to a register number in ARG2 and a bit number in ARG3. It also checks that ARG2 is less than or equal to 37777_8 and that ARG3 is less than or equal to 17_8.

If the string contains illegal characters, or if the return values are illegal, STATUS will be nonzero. If the conversion was successful, STATUS will be returned as zero.

EXAMPLE

```
REG = 0
BIT = 0
INPUT"Enter point number";I$
CALL BASDRV(B.SRBCONV, STATUS, I$, REG, BIT, B)
IF STATUS<>0 THEN PRINT"Illegal Point Number":STOP
PRINT"Point decoded as "OCT$(REG)"."OCT$(BIT)
```

SRCONV --- Convert String to Register

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 48	Function Number	Integer
STATUS	Return Code	Integer
ARG1	String to convert	String
ARG2	Register return	Integer
ARG3		
ARG4		

DESCRIPTION

Converts an octal string to a binary register number returned in ARG2.

COMMENTS

If the string contains illegal characters, or if ARG2 is greater than 37777_8, STATUS will be returned as nonzero.

EXAMPLE

```
REG = 0
INPUT"Enter register number";I$
CALL BASDRV(B.SRCONV, STATUS, I$, REG, B, B)
IF STATUS<>0 THEN PRINT"Illegal register number.":STOP
PRINT"Register decoded as "OCT$(REG)
```

SETDLA --- Set Communications Wait

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 49	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New Delay	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

SETDLA allows you to control the length of time that BASDRV waits before returning with a "Timeout Error". You may need to adjust the delay if you are working with a complex RNET network or a large AutoMate program.

COMMENTS

The default delay is "1". Delay units correspond approximately to seconds. Therefore, values below 100 are recommended.

EXAMPLE

```
DEL = 50
CALL BASDRV(B.SETDLA, STATUS, DEL, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

WORDAR --- Unpack Word into Array

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 50	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Integer to unpack	Integer
ARG2	Target array	Integer Array
ARG3		
ARG4		

DESCRIPTION

This command splits an integer into its 16 component bits. It stores the bits in the first sixteen elements of the target array. Bit 0 (the least significant bit) is assigned to element 0 of the array.

EXAMPLE

```
REG = &O2000
REGCT = 1
REGVAL = 0
DIM BAR(16)
CALL BASDRV(B.RDREG, STATUS, REG, REGCT, REGVAL, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
CALL BASDRV(B.WORDAR, STATUS, REGVAL, BAR(0), B, B)
PRINT"Register 2000's value in binary: ";
FOR LA=15 TO 0 STEP -1
  IF BAR(LA) THEN PRINT"1"; ELSE PRINT"0";
  NEXT LA:PRINT
```

ARWORD --- Pack Array into Integer

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 51	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Source Array	Integer Array
ARG2	Target Integer	Integer
ARG3		
ARG4		

DESCRIPTION

This function is the converse of WORDAR. It packs the first sixteen elements of the source array ARG1 into the destination integer ARG2. Element 0 of ARG1 determines the status of Bit 0 of the destination integer.

COMMENTS

ARWORD checks each of the first 16 elements of the source array in turn. If the element is nonzero, that bit of the target integer will be set. If the element is zero, the target bit will be cleared.

EXAMPLE

Assume array BAR(16) has been initialized to the desired bit pattern...

```
REG = &O2000
REGCT = 1
REGVAL = 0
CALL BASDRV(B.ARWORD, STATUS, BAR(0), REGVAL, B, B)
CALL BASDRV(B.WRREG, STATUS, REGVAL, REGCT, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Register 2000 set to"REGVAL
```

SETOND --- Set Origin Node Number

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 52	Function Number	Integer
STATUS	Return Code	Integer
ARG1	New Origin Node	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sets the origin node number. This is the node that the PC transmits from.

COMMENTS

The node number must be less than 255.

EXAMPLE

```
ONODE = 4
CALL BASDRV(B.SETOND, STATUS, ONODE, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"PC node set to"ONODE
```

SETMSK --- Set Interrupt Mask

*** DISCONTINUED IN V2.0 ***

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 53	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Mask Value	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Prior to Version 2.0, this function provided a way for the user to mask out interrupts that were disturbing communications with the AutoMate. The advent of interrupt-deriven reception in Version 2 removed the need for this function.

In order to maintain compatibility with previous versions, Version 2.0 of the AutoMate^{tm} Driver will accept calls to this function, but it always returns a STATUS value of zero and does nothing.

STPORT --- Set Communications Port

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 54	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Port Number	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Selects the communications port. By default, port COM1: is used.

COMMENTS

ARG1 can be 1 (COM1), 2 (COM2), 3 (COM3), or 4 (COM4). If you have trouble using ports COM3 or COM4, you may need to set up the I/O Port Base Address. Please refer to PSYST on page 95 for more information.

EXAMPLE

```
PORTNO = 2
CALL BASDRV(B.STPORT, STATUS, PORTNO, B, B, B)
PRINT"Communications Port Set to COM2"
```

RDCHEK --- Read AutoMate Checksums

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 55	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Application Checksum	Integer
ARG2	20K Register Checksum	Integer
ARG3	I/O Config. Checksum	Integer
ARG4		

DESCRIPTION

The AutoMate 20, 30, and 40 processors maintain three one-byte checksums of important memory regions. These checksums can be used to monitor the integrity of a program running on the AutoMate, since any modification of the application program or I/O configuration will disturb the checksum.

COMMENTS

After this call, the current checksum values will be returned in the arguments as shown. Since the checksums are byte values, the integers returned will always be between 0 and 255.

EXAMPLE

```
APPC = 0 : CC = 0 : K20C = 0
CALL BASDRV(B.RDCHEK, STATUS, APPC, K20C, CC, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT "Current checksums are:"APPC",".K20C", and"CC
```

STEXT --- Send Text to Serial Port

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 56	Function Number	Integer
STATUS	Return Code	Integer
ARG1	String to send	String
ARG2		
ARG3		
ARG4		

DESCRIPTION

Sends an arbitrary text string to the serial port at the current baud rate. Strings are always sent with one stop bit and no parity.

COMMENTS

You may wish to use this command to send dialing strings to a modem. Any reply from the destination device will be lost. STATUS will return -1 if the string could not be transmitted for some reason.

EXAMPLE

```
STR = "AT D 1 800 555 1212" + CHR$(13)
CALL BASDRV(B.STEXT, STATUS, STR, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP ELSE
    PRINT"Dialing..."
```

FMO2MS --- Motorola Floating Point to Microsoft

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 57	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Count	Integer
ARG2	Input Array (Motorola)	Long Integer Array
ARG3	Output Array (Microsoft)	Floating Point Array
ARG4		

DESCRIPTION

The floating point format used by the AutoMate, called the Motorola Fast Floating Point format, is not compatible with the real number types used on the PC. In order to read or write floating point numbers on the AutoMate, you must first translate them to (or from) the PC's native format.

The FMO2MS command converts COUNT numbers in the AutoMate's format (Input Array) to the PC's format (Output Array, single precision integer). Remember that AutoMate floating point numbers are 4 bytes long.

EXAMPLE

```
' Display 8 floating point numbers stored at 2000-2017
DIM MOTAR(8) AS LONG
DIM MSAR(8) AS SINGLE
REG = &O2000 : COUNT = 16
CALL BASDRV(B.RDREG, STATUS, REG, COUNT, MOTAR, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
COUNT = 8
CALL BASDRV(B.FMO2MS, STATUS, MOTAR, COUNT, MSAR, B)
FOR L=0 TO COUNT-1:PRINT USING "#.###"; MSAR(L):NEXT L
```

FMS2MO --- Microsoft Floating Point to Motorola

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 58	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Count	Integer
ARG2	Input Array (Microsoft)	Floating Point Array
ARG3	Output Array (Motorola)	Longword Array
ARG4		

DESCRIPTION

The floating point format used by the AutoMate, called the Motorola Fast Floating Point format, is not compatible with the real number types used on the PC. In order to read or write floating point numbers on the AutoMate, you must first translate them to (or from) the PC's native format.

The FMS2MO command converts COUNT numbers in the PC's format (Input Array) to the AutoMate's format (Output Array, long integer). Remember that AutoMate floating point numbers are 4 bytes long.

EXAMPLE

```
' Write 8 floating point numbers to 2000-2017
DIM MOTAR(8) AS LONG
DIM MSAR(8) AS SINGLE
COUNT = 8
FOR L=0 TO COUNT-1:MSAR(L)=L*3.14159:NEXT L
CALL BASDRV(B.FMS2MO, STATUS, MSAR, COUNT, MOTAR, B)
REG = &O2000 : COUNT = 16
CALL BASDRV(B.WRREG, STATUS, REG, COUNT, MOTAR, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

FRMPRO --- Frame Protect Options

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 59	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Protect Mask	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Not valid for Windows DLLs.

The Frame Protect command is intended for use with multitasking operating environments like DesqView™. It will attempt to protect the time-critical portions of the communications cycle from interruption by other tasks.

Currently, the only frame protect options allowed are:

ARG1	**Protection**
1	DesqView Protection On
0	No Protection.

In Version 2.0, the Frame Protect command only operates with DesqView™. It will have no effect on other operating environments.

If you are using DesqView™ and notice frequent Timeout or Checksum errors, you should try enabling Frame Protect.

EXAMPLE

```
' Enable Protection under DesqView™
FLAG = 1
FOR L=0 TO COUNT-1:MSAR(L)=L*3.14159:NEXT L
CALL BASDRV(B.FMS2MO, STATUS, MSAR, COUNT, MOTAR, B)
REG = &O2000 : COUNT = 16
CALL BASDRV(B.WRREG, STATUS, REG, COUNT, MOTAR, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

OFF --- Deactivate Driver

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 60	Function Number	Integer
STATUS	Return Code	Integer
ARG1		
ARG2		
ARG3		
ARG4		

DESCRIPTION

Version 2.0 of the Driver normally uses your computer's hardware interrupts to ensure reliable communications, even when running in a multitasking operating environment. It is **ABSOLUTELY ESSENTIAL** that you allow the Driver to deactivate these interrupts before you exit your application program. If you fail to do this, your computer may crash unpredictably, even long after you have left your program.

The OFF function deactivates the hardware interrupts and deactivates the communications port that was in use by the Driver. You should call the Driver with OFF as the function before your program returns to DOS. C programmers can use the **atexit** function to make sure OFF gets called; users of other languages must perform the call as part of their exit routines.

The OFF function is available both as a normal BASDRV call, and through its own entry point. To use the direct entry, execute a CALL BASDRVOFF. Either method will safely shut down the Driver.

EXAMPLE

```
' End of program.  Shut down Driver before leaving
CALL BASDRV(B.OFF, STATUS, B, B, B, B)
END
```

PSYST --- Set Port System Parameters

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 61	Function Number	Integer
STATUS	Return Code	Integer
ARG1	I/O Port Base Address	Integer
ARG2	Transmit Mode	Integer
ARG3	Receive Mode	Integer
ARG4	Interrupt Chaining Enable	Integer

DESCRIPTION

Not valid for Windows DLLs.

The PSYST command is used to manipulate certain low-level operating parameters. Most users will never need to call PSYST. If you do need PSYST, use it with **GREAT CAUTION.** Incorrect parameters to PSYST call cause your system to lock up or crash.

The first parameter, ARG1, tells the Driver the address of your serial communications port. Normally, it can find out this information from the computer's BIOS, but there are some situations where you may need to set the Port Address manually.

Certain computers cannot automatically detect all of their serial ports. This usually happens when there is a "gap" in the port sequence (e.g., you have COM1 and COM3 but no COM2). The Driver can use these "missing" ports to communicate, as long as you use the PSYST command to tell it where the ports are located.

The standard serial port addresses are:

Port	I/O Base Addr (Hex)	Interrupt
COM1	3F8	IRQ4
COM2	2F8	IRQ3
COM3	3E8	IRQ4
COM4	2E8	IRQ3

You should use the correct address from the table above, unless you are sure that your communications port is located elsewhere.

The second and third arguments control the strategy used by the Driver to actually transmit and receive data. The defaults should be acceptable on most systems, but you may need to change modes if you have difficulty communicating or if you experience unexplained system crashes.

The legal modes for ARG2 (Transmit Strategy) and ARG3 (Receive Strategy) are:

Mode	Strategy
0	**BIOS.** The Driver uses BIOS calls to perform I/O. Slowest but most widely compatible mode.
1	**Direct.** The Driver manipulates the communications port hardware directly using I/O Ports. Fastest transmit strategy, acceptable receive strategy.
2	**Interrupt Driven.** Hardware interrupts pace I/O. Best strategy for reliable reception.

The default strategies are **1** (Direct) for Transmission and **2** (Interrupt Driven) for Reception. These modes should be the best selection for most situations.

We do not recommend that you use Interrupt Driven Transmission unless it is absolutely necessary. This mode is slower than Direct Transmission, and it will usually degrade system performance significantly: an interrupt will be generated for each outgoing *and* each incoming character.

Interrupt Driven Reception, however, is recommended unless prohibited by your system configuration. Since the AutoMate^{tm} does not provide hardware handshaking to pace its transmissions, this is the only mode that can ensure the proper capture of all incoming data bytes.

The last PSYST parameter, ARG4, controls Interrupt Chaining. Normally, once the Driver has serviced an interrupt, it returns control directly to the interrupted process. In some circumstances, however, it may be desirable to pass control to the interrupt handler that previously serviced the interrupt in question. If you set ARG4 to a non-zero value, the Driver will "chain" control to the previous handler once it has completed its own service routine. Note, however, that the serial port will already have been serviced once control is passed; this may confuse some handlers.

If you don't understand the previous paragraph, don't worry. Just pass PSYST a zero value for interrupt chaining and skip ahead to the Example; zero (No Chaining) is the default value and should be used unless there is a particular reason to activate chaining.

Shared Interrupts. There is one instance where the Driver will chain to another handler regardless of the ARG4 setting. If an interrupt is received on the current interrupt line that was *not* generated by the current serial port controller, the Driver will pass control to the previously existing handler. It will perform no service.

This automatic chaining is needed for systems which have shared communications interrupts in simultaneous use. For example, COM1 and COM3 share the same interrupt signal line, IRQ4. If both of these ports are in use simultaneously, each handler must pass control to the other if the interrupt is not "for them." The Driver supports this type of sharing, though many other software packages do not.

EXAMPLE

```
' Select COM3 with normal I/O strategies
PORTN = 3
BADDR = &H3E8 : XMODE = 1 : RMODE = 2 : CHMODE = 0
' Switch to COM3
CALL BASDRV(B.STPORT, STATUS, PORTN, B, B, B)
' Set base address
CALL BASDRV(B.PSYST, STATUS, BADDR, XMODE, RMODE, CHMODE)
```

KEYPORT --- Set Hardware Key port

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 63	Function Number	Integer
STATUS	Return Code	Integer
ARG1	Hardware Key Port	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

By default, the Driver assumes that the Hardware Key is located on LPT1. However, you can tell the Driver to look for the Key on another parallel printer port with this command. ARG1 selects the port where the Key is located and can range from 1 to 3.

When you execute the KEYPORT command, the Driver will immediately try to locate the Key on the new port. If the Key is not found, the Driver will return a STATUS value of -2.

Note. This command is not needed for Windows DLL Drivers or the Incorporated Versions of DOS Drivers; those products have no copy protection.

EXAMPLE

```
' Select LPT2 for Hardware Key
PORTN = 2
CALL BASDRV(B.KEYPORT, STATUS, PORTN, B, B, B)
IF STATUS=-2 THEN PRINT"Hardware Key Not Detected.":STOP
```

PCLINK --- Set Up R-Net PC Link

SYNOPSIS

CALL BASDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 64	Function Number	Integer
STATUS	Return Code	Integer
ARG1	I/O Port Address	Integer
ARG2	PC Link Node Number	Integer
ARG3	PC Link Max Node Number	Integer
ARG4		

DESCRIPTION

The Reliance R-Net PC Link card is an adaptor that can connect your computer directly to R-Net. When using the PC Link, your computer becomes a node on the R-Net without the need for a Gateway.

Executing the PCLINK command will cause the Driver to initialize the PC Link card. If the initialization is sucessful, all further Driver communications calls will be processed via the PC Link, not the serial port.

Before executing the PCLINK command, you must know the "I/O Port Base Address" that has been selected for the PC Link card. This address is set using DIP switches on the card. The factory-selected address is 250 Hex.

Note. You *must* start up the PC Link card before you can communicate with it using the Driver. This is done using the RTINST utility supplied with the PC Link. RTINST will load the PC Link's executive program onto the card and start it up.

If you do not start up the card before starting your application that uses the Driver, the PCLINK command will fail because the PC Link will appear to be "not there." The PC Link will not respond to requests from the Driver until it has been properly started up.

If you are communicating via the PC Link, calling PCLINK with ARG1 (the I/O Port Address) equal to zero will disconnect the PC Link card and return the Driver to serial operation. Executing the OFF command will also disconnect the PC Link.

Using the PCLINK command will automatically set the correct source node (the equivalent of executing a SETOND command with the Node Number argument equal to ARG2). To communicate with a particular processor, you will still need to select the destination node and slot of the target using the SETNOD and SETDSLT commands respectively.

Note. If you are using a 386 memory manager like HIMEM / EMM386, 386MAX, OR QEMM, you need to take special care while using the PC Link Card. It is "invisible" when your computer is first powered on, and only becomes operational after being started by the RTINST utility.

Accordingly, the card will not be automatically detected by most memory managers. These programs will "map" memory over the PC Link's shared memory area, causing the computer to malfunction.

You will need to configure your memory manager to "exclude" the 16KB region indicated when you started the card with the RTINST utility. This is especially important if you plan to use the Driver DLL under Microsoft Windows. If the PC Link has not yet been initialized when Windows is started, the DLL will be unable to communicate with the card.

Consult your memory manager's documentation for more information on "excluding" addresses. For EMM386.EXE, if you started the card with a memory address of D000, you would add the text "X=D000-D3FF" to your EMM386 command line in the config.sys file.

EXAMPLE

```
'
' Connect via R-Net PC Link at base address 250 Hex with
'  node number 2 and max node number 5
BASE = &H250
PORTN = 2
MAXN = 5
CALL BASDRV(B.PCLINK, STATUS, BASE, PORTN, MAXN, B)
IF STATUS=0 THEN PRINT"Communicating via R-Net PC Link" ELSE
  PRINT"Cannot connect with PC Link, error "STATUS:STOP
```