MODBUS™ Communications Driver

User's Manual

Version 2.100 -- March 21, 1993

SOFTWARE LICENSE AGREEMENT

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 MODBUS driver (Incorporated Version), this license is amended to provide for the free or for-profit distribution of software incorporating MODBUS Driver code as follows: you may distribute executable programs containing the complete and unaltered ACS MODBUS 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 2.1

There have been several important changes in the operation of the MODBUS 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 MBDRV
```

will load the driver automatically. There is no longer any need to load the MBDRV.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 MBDRV with the function synopsis:

```
CALL MBDRV(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 50.

Support for “software key” protection has been discontinued.

Introduction

The MBDRV Communications Driver is a memory resident DOS extension written in 8086 machine language which enables BASIC programs to communicate with devices that understand the Gould MODBUS™ Process Control Protocol.

The driver provides an easy way for the user to develop programs that access a MODBUS device's points and registers. Information is passed using standard variables in the user's host language; the MODBUS driver handles all protocol formatting and variable conversion in both directions. MBDRV always operates in "RTU" (Binary) mode.

The MODBUS 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 individual function synopses.

General Information

All of the language interfaces (called "bindings") share the same general operating principles. First, there is always a memory-resident portion of the driver that must be loaded before you can access the MODBUS 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 Microsoft QuickBASIC 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.

"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 MBDRV) as time permits.

If you find a bug in MBDRV, 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 an IBM 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.

The rest of this manual consists of language interface descriptions and synopses of MBDRV's functions. In the synopses, all variables are integers unless otherwise noted. The variable B is used as a dummy placeholder to ensure that six parameters are always passed to MBDRV.

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.

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 MBDRV 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

The single-user Driver is protected by a 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 periodically 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 program will not operate.

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 MBDRV with the function synopsis:

```
CALL MBDRV(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 50) 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 IBM PC or compatible to the MODBUS device.

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 IBM PC serial port is a DB25M (25-pin Male) connector. Here are its pinouts (pins not listed 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 IBM 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	IBM PC Pin	IBM AT Pin	Device Pin
1	Ground	7	5	7
3	TD	2	3	3
4	RD	3	2	2

Unfortunately, not all MODBUS devices have standard serial ports. You may need to experiment in order to find the correct cabling setup. A “breakout box” or similar device can be very helpful while trying to set up a serial communications link.

Interfaces

Interpreted BASIC

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

MBDRV ↵Enter

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

In the Interpreted (IBM) BASIC interface, information is passed to and from the processor via normal integer and string variables. The BASIC "MBDRV" statement provides access to the communication functions.

MBDRV communicates with the MODBUS device via your IBM PC's serial port (COM1). You will need the cable provided by ACS to connect the device to the computer.

To use the 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. Note that MBDRV does not consume any of BASIC's data or program space since it is loaded *before* BASIC.

The driver is provided in a file called MBDRV.EXE. To load the driver into memory, type:

MBDRV ↵Enter

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 or turn off the computer.

In order to call the Driver from BASIC, you must know where in memory it is located. When you load the driver, it stores its location (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 ]
MBSEG = PEEK(&H4F2)+PEEK(&H4F3)*256  [ Read mbdrv seg. ]
DEF SEG = MBSEG [ Establish access to driver ]
MBDRV = 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 MBDRV merely sets the destination address within the segment to 0.

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

```
CALL MBDRV(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 driver functions readily accessible is to use a fixed length parameter block and a series of function codes.

The MBDRV 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 MBDRV. Also, you should be very careful about the types of variables in the MBDRV. 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 MBDRV 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 CALL MBDRV 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 twenty standard functions provided by the driver.

The second parameter (STATUS in the sample call) is an integer 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 probably check the status codes after each call.

The third, fourth, fifth, and sixth parameters vary according to the function being called. Again, you *must always* pass a total of 6 parameters to the Driver!

Arrays

When using Arrays with the Interpreted BASIC binding, always pass them by their first element. In other words, if you have DIMensioned an array called INTAR with 10 elements, and you wish to refer to it in a MBDRV call, you should type INTAR(0). This will take care of problems sometimes caused by BASIC, which considers "INTAR" a legal variable name distinct from INTAR the array.

The correct way to call MBDRV with the array argument is shown in this code fragment (all variables are integers):

```
DIM INTAR(5)
REG = 40300
CT = 3
CALL MBDRV(MB.RDOR,STATUS,REG,CT,INTAR(0))
```

Sample Files

Two BASIC program files are provided on the distribution diskette to help you use MBDRV. They are named MBHEADER.BAS and MBSHORT.BAS. MBHEADER.BAS contains a commented header which takes care of setup for MBDRV calls. It declares variables beginning with 'M' to be integers and assigns function numbers to named variables to make calling MBDRV functions easier. For example, function number 1 (RDIS, Read Input Status) is assigned to the variable MB.RDIS. MBHEADER.BAS also performs the standard call to establish communications.

MBSHORT.BAS is a version of MBHEADER.BAS which has all of the comments removed.

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

There is another program file, called MBDEMO.BAS, on the distribution diskette. This program is intended as a sample of what can be done with MBDRV. To use it, issue the following commands at DOS command level:

```
MBDRV ↵Enter
BASIC MBDEMO ↵Enter
```

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

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

QuickBASIC V4.5

Important Note! The QuickBASIC 4 Driver is **NOT** compatible with the BASIC interpreter, nor is it compatible with QuickBASIC prior to V4. This Driver can **ONLY** be used with QuickBASIC Version 4 or later.

When entering QuickBASIC, you must load the Driver when you load QB. To do this, make sure that the Quick Library file MBQB4DRV.QLB, supplied with the binding, is in the current directory. Then type:

```
QB <BASIC file> /L MBQB4DRV ↵Enter
```

This command loads QuickBASIC with the binding resident. If you do not load the binding, you will get "Unresolved Subprogram Reference" errors.

To load the driver and QuickBASIC with the supplied QuickBASIC demo program MBQBDEMO, type the following commands:

```
QB MBQBDEMO /L MBQB4DRV ↵Enter
```

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

Press ⇧Shift + F5 to run the demo. Make sure that you have connected to the MODBUS device before starting the program.

Arrays

When using Arrays with the QuickBASIC V4 binding, always pass them by their first element. In other words, if you have DIMensioned an array called INTAR with 10 elements, and you wish to refer to it in a MBDRV call, you should type INTAR(0). If you always use this technique, you will avoid problems caused by QuickBASIC moving an array between calls without updating the Array Descriptor. If you specify INTAR(0), the Driver will always get what it expects, a pointer to the first element of the destination array.

The correct way to call MBDRV with the array argument is shown in this code fragment (all variables are integers):

```
DIM INTAR(5)
REG = 40300
CT = 3
CALL MBDRV(mbRDOR,STATUS,REG,CT,INTAR(0))
```

DECLARE & Header file

For your convenience, we have included a "Header" include file which defines all of the MBQB4DRV function numbers as CONSTants. This file is called MBDRV.H. You may wish to use the $INCLUDE metacommand to bring this file into your BASIC Driver programs.

In this header file, the MBDRV function numbers are defined as integer constants beginning with "mb". For example, the constant mbWRSC is defined as the integer 5, which corresponds to function WRSC (Write Single Coil).

The file also includes the statement:

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

This statement **MUST** be included in all QB 4 programs that call the Driver. It will cause QuickBASIC to send segmented addresses to the Driver, in addition to making sure that you always call MBDRV with the proper number of parameters.

Technically, the DECLARE statement forces QB to pass all MBDRV parameters with segmented addresses. QB 4 may or may not choose to place arrays and variables in "near" memory, depending on a complex (almost erratic) set of criteria which cannot be tested by the Driver. Accordingly, the QB 4 Driver must use "far" (segmented) addressing, which is less efficient.

Passing Near (unsegmented) parameters to the QB 4 Driver is guaranteed to be spectacularly fatal!

You can achieve the same result as the DECLARE statement by changing all of the "CALL MBDRV" statements in your program to "CALLS MBDRV", but this will not count call parameters like the DECLARE statement does.

Array Note

IMPORTANT! For commands that take multiple arrays (BRKRCV, SNDUSR, etc.) as parameters, it is *crucial* that the arrays be in the same 80x86 data segment. This can be a problem in QuickBASIC, which often places data in multiple data segments.

Since QuickBASIC does not allow you to control segment allocation, you should take precautions. For example, you could copy the array contents into separate vectors of a multi-dimensional array (such arrays are always kept in a single segment).

Turbo PASCAL V7.0

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 mbdrv 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 checking 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. In other words, you can use a constant for the function number, something you could not do with BASIC (you had to assign the function number to a variable before the call).

A sample Turbo PASCAL program called MBDTEST.PAS is also supplied. This program permits you to monitor a table of registers on the MODBUS device. To run it, load the Turbo PASCAL Driver, load Turbo PASCAL with MBDTEST as the current file, and press Alt-R to Run the sample. The program should connect to the MODBUS device and ask you for a starting register and a length. Once you have supplied these parameters, the program will run until you press a key.

The MBPASDRV Unit

The Turbo 4 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 MBPASDRV;
```

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

The MBPASDRV "unit" contains a list constant function numbers used to access MBDRV functions. All of these constants begin with "MB_". For example, the constant MB_RDOR is defined in the unit as 3, the function number for Read Output Register. The file MBPASDRV.I is included so that you can see the list of constants found in the unit file.

Array Note

IMPORTANT! For commands that take multiple arrays (BRKRCV, SNDUSR, etc.) as parameters, it is *crucial* that the arrays be in the same 80x86 data segment. This can be a problem in Turbo Pascal, which often places data in multiple data segments.

Since Turbo Pascal does not allow you to control segment allocation, you should take precautions. For example, you could copy the array contents into separate vectors of a multi-dimensional array (such arrays are always kept in a single segment).

C

The Microsoft C binding is very similar to the standard BASIC interface. Calls to the driver take the form:

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

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

The function mbdrv() should be declared as:

```
extern int mbdrv();
```

Note: If you are using C++, you must use the syntax:

```
extern "C" int mbdrv();
```

The driver interface is found in a library file called MBDRV.LIB. This file contains the actual mbdrv() function, and you must link it with any C program that calls the driver. MBDRV is designed for use with the Small memory model. A second library file, MBDRVM, is provided for use with the Medium memory model.

Note! The standard binding has no provision for operation with the Compact, Large, or Huge models. Support for Compact/Large model programs is available by special order. Please see the "Large Model" section below for more information on working with Large model programs.

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.

The MBDRV C binding 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.

Note! If you intend to use MBDRV calls that contain constant "value" parameters, be very careful about C's automatic typing. For example, if you refer to Register 40137, C will pass this as a **long** by default, since it is greater than 32767. For proper operation, you must "cast" the constant to **unsigned int** like this:

```
st = mbdrv(MB_RDOR, 1, (unsigned int) 40137, 2, intar);
```

This command reads 2 Holding registers starting at 40137 into the integer array intar.

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 mbdrv() if it is not needed. Any nonzero status value indicates an error has occurred.

Further, while BASIC and PASCAL cannot handle variable-length parameter lists, C can. 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, MBMSCDRV.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 "MB_name". For example, symbol MB_WRMR is assigned to 9, the function number for the WRMR (Write Multiple Registers) call. To use this file, simply put the statement "#include <mbmscdrv.h>" near the top of your C source file.

Large Model

If you ordered the Large model MODBUS driver, there are several important facts to keep in mind. Basic operation of the Driver is unchanged, but ***all parameters must be passed by reference***. The Small model Driver relies on the fact that scalar data types and pointers have the same size (16 bits) in the Small model. Large model pointers are 32 bits wide, so the Driver can no longer accept parameter lists containing both pointers and word-length parameters.

Since all Driver parameters must be passed by reference, you must assign numeric parameters to variables before passing them to the Driver. The example above becomes:

```
int addr, count, intar[10], st;
unsigned int streg;

addr = 1;
streg = 40137;
count = 2;

st = mbdrv(MB_RDOR, &addr, &streg, &count, intar);
```

Incorporated Version

The Incorporated Driver for Microsoft C is supplied as a LIBrary file, MBMSCI.LIB (MBMSCIL.LIB for the Large model). You should include this library name in your Linker command. With the Incorporated Driver, there is no resident portion to load; the entire Driver is included in the library.

Error Codes

Error	Code
Buffer Overflow or Illegal Return Frame	-4
No Such Port (CHNPOR)	-3
Driver Not Resident	-2
Timeout	-1
No Error	0
Checksum Error	1
Illegal Parameter	2
Illegal Command Code	101
Illegal Address	102
Illegal Data Value	103
Command Error	104

These codes are returned in the STATUS variable upon completion of the call. Any nonzero value indicates an error condition.

MODBUS error return codes come back with 100 decimal added to them (see errors 101 - 104 above). MBDRV supports error codes greater than 4 (if your device's implementation of the MODBUS protocol uses them) in the same way.

MODBUS Communications Driver - Function Quick Reference

Function Quick Reference by Function Number

Function Name	Function Number	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Description
RDOS	1	ADDR	STON	COUNT	&DAR	Read Output Status
RDIS	2	ADDR	STIN	COUNT	&DAR	Read Input Status
RDOR	3	ADDR	STOR	COUNT	&DAR	Read Output Registers
RDIR	4	ADDR	STIR	COUNT	&DIR	Read Input Registers
WRSC	5	ADDR	STON	DATA		Write Single Coil
WRSR	6	ADDR	STOR	DATA		Write Single Register
LOPBAK	7	ADDR	TDATA	RDATA		Loopback Test
WRMP	8	ADDR	STON	COUNT	DATAR	Write Multiple Points
WRMR	9	ADDR	STOR	COUNT	DATAR	Write Multiple Registers
SNDUSR	10	SNDAR	CTLAR			Send User Command
SETDEL	11	LNGDEL	QUIDEL	DLMUL		Set Communications Delays
CHNPOR	12	PORT	BAUD			Set Communications Parameters
RCVUSR	13	&ADDR	&FC	&RCVL	&RDATA	Receive Incoming Command
BRKRCV	14	RCVL	CTLAR	RDATA	&DATAR	Dissect Incoming Frame
SRBCONV	15	STR	&REGN	&BITN		Convert String to Register and Bit
SRCONV	16	STR	&REGN			Convert String to Register

MODBUS Communications Driver - Function Quick Reference

WORDAR	17	DATA	&DATAR		Break Word into Array
ARWORD	18	DATAR	&DATA		Assemble Array into Word
STEXT	19	STR			Send Text to Serial Port
IOMAP	20	FLAG			Control I/O Mapping
OPPAR	21	PARITY	STOPS	DATAB	Other Comm. Port Parameters
KEYPORT	22	PORTN			Select Hardware Key port

MODBUS Communications Driver - Function QRF

Variable Names used in Quick Reference Table

Symbol	Description
&	Variable may be modified by function. Must be passed by reference.
ADDR	Destination Device Address
BAUD	Integer Baud Rate Code (0 - 7)
CTLAR	Integer “control” array
COUNT	Integer count
DAR	Integer “destination” array
DATA	Integer value storage
DATAB	Integer number of Data Bits (7 or 8)
DATAR	Integer data array
DLMUL	Integer Delay Multiplier
FC	Integer Function Code
FLAG	Integer Flag
INTAR	Integer array
INTVAL	Integer variable
LNGDEL	Integer “Long” delay
OUTSTR	String to transmit
PARITY	Integer Parity flag (0 = None, 1 = Odd, or 2 = Even)
PORT	Integer Comm. Port Number (1 or 2)
QUIDEL	Integer “Quiet” delay
RCVAR	Integer “receive” array
RCVL	Integer “received” length
RDATA	Integer “returned data” array

SNDAR	Integer “send” array
SRAR	Integer “source” array
STIN	Starting Input Number (3*xxxx*)
STON	Starting Output Number (0*xxxx*)
STOPS	Integer Number of Stop Bits (1 or 2)
STIR	Starting Input Register Number (3*xxxx*)
STOR	Starting Output Register Number (4*xxxx*)
STR	String variable
TDATA	Integer loopback transmit value

Function Summary

This section contains a "synopsis" of each function supported by the MODBUS driver. The first nine functions are simple "transaction" functions that issue a command to the MODBUS device and return an answer. They are used to read and write data to the MODBUS device.

The remaining functions are "support" routines used to configure the MODBUS Driver, send and receive user-generated commands, and perform conversions.

As mentioned above, each MBDRV function is shown in BASIC format. However, only the form of the call will change when the Driver is used with other languages.

The MODBUS driver uses a total of four different variable types: Integer, Integer Array, Unsigned Integer, 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. Please consult the Language Interfaces section for more information on this topic.

Note: Throughout this section of the manual, we will refer to "addresses". These addresses are used as defined in the MODBUS Process Control protocol manual, that is, they are decimal numbers between 00000 and 49999. The corresponding addresses are:

Address	Writable	Type
0*xxxx*	Yes	Internal Discretes (digital points) and Discrete Outputs (Coils)
1*xxxx*	No	Discrete (digital) Inputs
3*xxxx*	No	Input Registers
4*xxxx*	Yes	Holding Registers

RDOS --- Read Output Status

SYNOPSIS

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

FNO = 1	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Dest. Device Address	Integer
ARG2	Starting Coil Number	Unsigned Integer
ARG3	Number of Coils to Read	Integer < 1950
ARG4	Return Array	Integer Array

DESCRIPTION

This function reads the status of Output Coils (address 0*xxxx*) on the MODBUS device. The coil status values are returned in a "packed" format, with each bit in the return array corresponding to one output coil. If the number of coils requested is not evenly divisible by 16, unused return array bits will be set to 0.

As a convenience, RDOS always stores the total number of words returned as the first element of the ARG4 array.

The MODBUS Driver allows you to read up to 1950 bits in one operation, but your MODBUS device may have a lower transaction length limit.

EXAMPLE

```
DIM INTAR(5)
REG = 20
CT = 23
CALL MBDRV(MB.RDOS, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

This example reads a total of 23 bits starting at Output Coil Number 20. In the return array INTAR, element 0 will contain 2, the number of words returned. Element 1 will contain the current values of coils 20 - 35, with Bit 0 (the LSb) corresponding to Coil 20 and Bit 15 (the MSb) corresponding to Coil 35. Element 2 of INTAR will contain Coils 36 - 42, with Bit 0 for Coil 36, Bit 7 for Coil 42, and Bits 8 - 15 set to 0.

RDIS --- Read Input Status

SYNOPSIS

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

FNO = 2	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Starting Input No.	Unsigned Integer
ARG3	No. of Inputs to Read	Integer < 1950
ARG4	Return Array	Integer Array

DESCRIPTION

This routine is similar to RDOS (Read Output Status), except it reads the status of Input points (address like 1*xxxx*). The Input values will be packed 16 bits per return array element, just as in the Read Output command.

Like RDOS, RDIS always stores the total number of words returned as the first element of the ARG4 array.

The MODBUS Driver allows you to read up to 1950 bits in one operation, but your MODBUS device may have a lower transaction length limit.

EXAMPLE

```
DIM INTAR(5)
REG = 10020
CT = 23
CALL MBDRV(MB.RDIS, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

RDOR --- Read Output Registers

SYNOPSIS

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

FNO = 3	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Starting Register No.	Unsigned Integer
ARG3	No. of Registers	Integer < 120
ARG4	Return Array	Integer array

DESCRIPTION

This command returns the values of Output registers (address 4*xxxx*), one register per return array element. The Driver permits you to read up to 120 registers in one operation, though your MODBUS device may require shorter requests.

EXAMPLE

```
DIM INTAR(5)
REG = 40115
CT = 3
CALL MBDRV(MB.RDOR, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
FOR L=0 TO 2 : PRINT REG+LA" = "INTAR(L) : NEXT L
```

RDIR --- Read Input Registers

SYNOPSIS

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

FNO = 4	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Starting Register No.	Unsigned Integer
ARG3	No. of Registers to Read	Integer < 120
ARG4	Return Array	Integer Array

DESCRIPTION

This command is analagous to the Read Output Registers command, except it returns the values of Input Registers, one register per return array element. The Driver permits you to read up to 120 registers in one operation, though your MODBUS device may require shorter requests.

EXAMPLE

```
DIM INTAR(5)
REG = 30227
CT = 3
CALL MBDRV(MB.RDIR, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
FOR L=0 TO 2 : PRINT REG+LA" = "INTAR(L) : NEXT L
```

WRSC --- Write Single Coil

SYNOPSIS

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

FNO = 5	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Coil Number	Integer
ARG3	New Value	Integer
ARG4		

DESCRIPTION

Writes a new value to a single Output coil (address 0*xxxx*). The Driver will set the Coil to 0 if "New Value" is zero, or to 1 otherwise.

EXAMPLE

```
CNO = 122
DAT = 1
CALL MBDRV(MB.WRSC, STATUS, CNO, DAT, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Coil 122 set to 1"
```

WRSR --- Write Single Register

SYNOPSIS

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

FNO = 6	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Target Register	Unsigned Integer
ARG3	New Value	Integer
ARG4		

DESCRIPTION

Ths command changes the value of a single Holding register (address 4*xxxx*) on the MODBUS device.

EXAMPLE

```
REG = 40116
DAT = 120
CALL MBDRV(MB.WRSR, STATUS, REG, DAT, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Register 40116 set to 120"
```

LOPBAK --- Loopback Test

SYNOPSIS

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

FNO = 7	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Transmit Data	Integer
ARG3	Returned Data	Integer Array
ARG4		

DESCRIPTION

This command performs a test to see if a MODBUS device is present and operating properly. To perform the Loopback test, send a known integer value as ARG2. If all goes well, the STATUS variable should be 0 on return. The Loopback test will also return two values in Integer Array ARG3 : Element 0 should always be 0, and Element 1 should contain the Integer value sent as ARG2.

Note that this command uses the "Mode 0" MODBUS loopback test. "Mode 1" is not supported.

EXAMPLE

```
DIM INTAR(3)
DAT = 1234
CALL MBDRV(MB.LOPBAK, STATUS, DAT, INTAR(0), B, B)
IF STATUS<>0 THEN PRINT"Loopback Communication Error":STOP
IF INTAR(0)<>0 OR INTAR(1)<>DAT THEN PRINT"Loopback failed."
```

WRMP --- Write Multiple Points

SYNOPSIS

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

FNO = 8	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Starting Point No.	Unsigned Integer
ARG3	Count	Integer < 1950
ARG4	Data	Integer Array

DESCRIPTION

This command is the converse of the Read Output Status command. It writes new values onto "Count" (ARG3) consecutive coils (address 0*xxxx*) starting at the Starting Point Number (ARG2). Like the Read Output Status command, the Data array, which contains the new values for the coils, is "packed". This is, each 16-bit word in the Data array corresponds to 16 Output coils, beginning with the LSb of array element 0 and continuing upwards toward Bit 15.

EXAMPLE

```
DIM INTAR(5)
CNO = 144
CT = 23
INTAR(0) = &H27CD : INTAR(1) = 127
CALL MBDRV(MB.WRMP, STATUS, CNO, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

This sample will write a total of 23 coil values. The least significant bit of INTAR(0) will determine the new value of Coil 144, Bit 1 of INTAR(0) corresponds to Coil 145, and so on, through Bit 7 of INTAR(1), which corresponds to Coil 166.

WRMR --- Write Multiple Registers

SYNOPSIS

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

FNO = 9	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Destination Address	Integer
ARG2	Starting Register No.	Unsigned Integer
ARG3	Count	Integer < 120
ARG4	Data	Integer Array

DESCRIPTION

Assigns values from ARG4 to *Count* consecutive Holding registers (address 4*xxxx*) starting with the register specified by ARG2. The Driver permits you to transmit up to 120 registers in one operation, but your MODBUS device may have a lower limit.

EXAMPLE

```
DIM INTAR(8)
REG = 40118
CT = 3
INTAR(0) = 10 : INTAR(1) = 20 : INTAR(2) = 30
CALL MBDRV(MB.WRMR, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Registers sent."
```

SNDUSR --- Send User Command

SYNOPSIS

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

FNO = 10	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Send Data	Integer Array
ARG2	Control List	Integer Array
ARG3		
ARG4		

DESCRIPTION

This command is designed to allow you to send commands not included in the basic set supplied with the Driver. This command handles the transmission half of the operation *only* (See the RCVUSR command for more information).

To use this command, you must initialize two arrays. The "Send Data" array contains the data words and bytes that make up the command. The "Control List" array determines the format of the string.

Each element of the Control array corresponds to one element of the Data array. When building the command string, the SNDUSR command examines the two lowest bits of each Control array element.

Control array Bit 0 is the Byte / Word Selector. If Bit 0 is zero, the corresponding element of the Data array will be appended to the transmission string as a Byte. If Bit 0 is 1, the Data element will be appended as a Word. Note that the Driver will automatically swap the high and low order bytes to convert the Word from Intel format to the MODBUS protocol format.

Control array Bit 1 is the "end of string" marker; when the SNDUSR command detects Bit 1 set, it appends the checksum word and sends the command string to the serial port.

The SNDUSR command will proceed through both Control and Data arrays one element at a time, adding Bytes or Words as specified until the End of String bit is detected in the Control array, or until the string length reaches the limit of 252 bytes. It will then transmit the command and return immediately to the user program. **SNDUSR does not wait for the reply!** You must use the RCVUSR command to retrieve the MODBUS device's answering string.

Note also that you must supply bytes for the entire string, including the Destination Address and Command Number.

You can also use SNDUSR to answer command strings originating *from* the MODBUS device (in MODBUS terminology, this is called "Slave mode"). Normally, the IBM PC would be the Master

device, originating commands and awaiting replies. However, you can operate in Slave mode using the RCVUSR command to await an incoming MODBUS command, then reply with the SNDUSR command.

> **Note:** SNDUSR examines only Bits 0 and 1 of the Control array elements. This can be important, because it allows you to place the SNDUSR and BRKRCV control masks into the same array for operations involving user-defined commands. SNDUSR never alters the Control array.

IMPORTANT! When using the SNDUSR command, it is *crucial* that the Send and Control Arrays be in the same 80x86 data segment. This is normally not a problem unless you are using a language like Quick Basic or Turbo Pascal, languages that often place data in multiple data segments.

If you are using a language that does not allow you to control segment allocation, you should take precautions. For example, you could copy the Control and Data array contents into separate vectors of a two-dimensional array (such arrays are always kept in a single segment). In Turbo Pascal, you can generally force the same segment by making sure that both arrays are either local or global variables.

EXAMPLE

```
DIM CTRL(10),DAT(10)
CTRL(0) = 0 : DAT(0) = &H11    ' Byte 0    : Address, Hex 11
CTRL(1) = 0 : DAT(1) = 2       ' Byte 1    : Function 2 (RIS)
CTRL(2) = 1 : DAT(2) = &HC4    ' Bytes 2,3 : Start, 196
CTRL(3) = 1 : DAT(3) = &H16    ' Bytes 4,5 : Length, 22
CTRL(4) = 2     ' End of String
CALL MBDRV(MB.SNDUSR, STATUS, B, DAT(0), CTRL(0), B)
IF STATUS<>0 THEN PRINT"Send --- "STATUS:STOP
ADDR = 0 : FC = 0 : RCVL = 0   ' Initialize variables that
   will be returned by RCVUSR
CALL MBDRV(MB.RCVUSR, STATUS, ADDR, FC, RCVL, DAT(0))
```

This example shows the procedure used to send command number 2, "Read Input Status". Of course, you would normally use the built-in function RDIS to do this, but it makes a good example. The equivalent RDIS call is:

```
DIM INTAR(5)
REG = 100197
CT = 22
CALL MBDRV(MB.RDIS, STATUS, REG, CT, INTAR(0), B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

First, the code fragment initializes the control and data arrays. The final string to be sent is: 11 02 00 C4 00 16 + checksum. The control array is therefore set to Byte (0), Byte (0), Word (1),

Word(1), End (2). This means that Elements 0 and 1 of the Data array will be formatted as a Bytes, and Elements 2 and 3 of the Data array will be formatted as words. As soon as Element 3 has been appended, the string will be sent.

After transmitting the request, the code fragment awaits a reply using the RCVUSR command. If all goes well, the MODBUS device's reply will be stored in the DAT array, along with the return address in ADDR, return Function Code in FC, and received length in RCVL.

Important Note. If you are planning to design your own command strings, it is *very important* to note the address mapping. Normally, this function is performed automatically by the Driver before the message is transmitted, but you must perform the same mapping for your own messages.

In this case, though the call is asking for values beginning at address 10197, the actual address word sent is 196 (decimal). Process Control Addresses are always sent with their "range" digit (1*xxxx* in this case) removed. Further, MODBUS internal addresses actually start from 0, although they are normally specified beginning at 1. Hence, location 10197 actually corresponds to internal location 196.

SETDEL --- Set Communications Delays

SYNOPSIS

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

FNO = 11	Function Number	Integer
STATUS	Error Return	Integer
ARG1	"Long" Delay	Integer
ARG2	"Quiet" Delay	Integer
ARG3	Delay Multiplier	Integer
ARG4		

DESCRIPTION

The MODBUS protocol does not have a specific packet-framing system. Instead, it relies on delays to indicate the beginning and end of transmissions. The SETDEL command allows you to adjust the delay periods used by MBDRV.

Long Delay. The "Long" Delay determines how long MBDRV will wait for a reply to a command request. If you send a command and the MODBUS device does not begin its reply before the Long Delay has elapsed, MBDRV will return the Timeout error code, -1. The Long Delay value defaults to 800, which is approximately equivalent to 5 seconds.

Quiet Delay. The "Quiet" Delay determines the "end of message" delay. Since the Process Control Protocol does not have a specific way of marking the end of a message, this delay is used to regulate the length of time MBDRV will wait before it decides that the end of the transmission has been reached. The default "Quiet" delay value is 10, which is approximately equal to 500ms.

In practice, when MBDRV is waiting for an incoming string, it begins by counting down the Long delay. When the first character arrives, the delay timer is reset to the "Quiet" delay. Each succeeding character restarts the timer with the "Quiet" delay. MBDRV detects the end of the message when the incoming line has been quiet for a long enough period to exhaust the Quiet delay.

Officially, the Process Control Protocol specifies that the Quiet delay can be as short as 4ms (for 9600 baud). Experience, however, has proven that delays that short may result in truncated messages. Since the Driver is designed to operate "one message and reply at a time", longer "Quiet" delays have no harmful effect except to slow MBDRV's response time slightly.

Delay Multiplier. The Delay Multiplier is a constant that determines the "slope" of the delay timer's response. Its default value of 800 (about 50ms per unit) should not normally be altered, unless you need to set the delay to an exact time value. Please call us if you need to set the delay timer to a specific time.

If you wish to change one or two of the delay constants without disturbing the others, simply pass -1 as the new value for the delay(s) that you do not wish to change.

Sadly, there is no hard and fast rule for the proper delay times to use. In fact, delay times can be one of the most frustrating aspects of the MODBUS protocol. Though the defaults should be adequate for most situations, you will probably wish to experiment to determine the best tradeoff between response speed and reliability.

EXAMPLE

```
NQD = 15 : DMY = -1
CALL MBDRV(MB.SETDEL, STATUS, DMY, NDQ, DMY, B)
```

This example sets the Quiet delay to 15, leaving the Long delay and Delay Multiplier unchanged.

CHNPOR --- Set Comm. Parameters

SYNOPSIS

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

FNO = 12	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Port Number	Integer
ARG2	Baud Rate Code	Integer
ARG3		
ARG4		

DESCRIPTION

Use this command to change the Port and Speed used by the Driver. ARG1, the port number, can be either 1 (for COM1:) or 2 (for COM2:). MBDRV MODBUS "RTU Mode" communications are always set for 8 data bits, 1 stop bit, no parity.

The Baud Rate code must be an integer from 0 to 7. Here is a table of the baud rate values:

Code	Baud Rate
0	110
1	150
2	300
3	600
4	1200
5	2400
6	4800
7	9600

EXAMPLE

```
NPR = 2
BAU = 6
CALL MBDRV(MB.CHNPOR, STATUS, B, NPR, BAU, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
```

```
PRINT"Port 2 Selected at 9600 baud." : PRINT
```

RCVUSR --- Receive Incoming Command

SYNOPSIS

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

FNO = 13	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Address Return	Integer
ARG2	Function Code Return	Integer
ARG3	Receive Length Return	Integer
ARG4	Data Return	Integer Array

DESCRIPTION

This command is normally used for one of two purposes: retrieving answers to commands sent via SNDUSR, or awaiting incoming commands in Slave Mode. It simply waits for and returns an incoming Process Control Protocol data frame. If no frame arrives within the time specified by the "Long" delay (see SETDEL), MBDRV will return a STATUS value of -1 (Timeout Error).

If a correctly-formatted string *does* arrive, MBDRV will pass back all the information needed to decode it. First, ARG1 will return the "Destination Address" specified in the incoming frame, ARG2 the Function Code, ARG3 the frame length in bytes, and ARG4 the frame itself. MBDRV also tests the frame's Checksum and returns a STATUS of 1 (Checksum Error) if the checksum is incorrect.

The Data Return array, ARG4, contains the incoming frame in packed bytes. In other words, the complete incoming string is saved in ARG4 starting at the LSB of ARG4(0). You will probably wish to use the BRKRCV command to unpack the command into bytes and words as appropriate. Since RCVUSR returns both the Destination Address and the Command Number, you can design different handlers for each Node and Command number.

Important Note. Make sure that the Data Return array ARG4 is dimensioned large enough to accomodate the longest incoming frame that you expect to receive. Generally, you should allow for up to 255 bytes (128 integer elements) of incoming data.

If you have sent a user-constructed command with the SNDUSR command and are using RCVUSR to retrieve the reply, be sure to call RCVUSR as quickly as possible after SNDUSR returns to ensure that no incoming bytes are lost.

If you are using RCVUSR to await command codes from a MODBUS "Master" device, you will probably wish to place the CALL in a loop that repeats until a legal incoming frame is detected.

EXAMPLE

```
DIM RCVAR(130)  ' Slave Mode example
ADDR = 0 : FC = 0 : RCVL = 0
STATUS = -1
WHILE STATUS = -1
    CALL MBDRV(MB.RCVUST, STATUS, ADDR, FC, RCVL, RCVAR(0))
    WEND
PRINT"Received command number"FC" addressed to node"ADDR
```

BRKRCV --- Dissect Incoming Frame

SYNOPSIS

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

FNO = 14	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Frame Length	Integer
ARG2	Control Array	Integer Array
ARG3	Frame Array	Integer Array
ARG4	Output Array	Integer Array

DESCRIPTION

The BRKRCV command is designed to break up the data in an incoming command or reply frame into its component parts. It basically performs the inverse of the SNDUSR command's assembly process. Like SNDUSR, BRKRCV processes all of the elements in the Frame Array one at a time, breaking them up into the elements of the Output Array according to the elements of the Control array.

Each element of the Control array corresponds to one element of the Output array. The Control array elements are interpreted as bit masks: Bit 2 is the Byte / Word selector, and Bit 3 is the End marker. MBDRV will proceed through the returned frame one byte or word at a time as indicated by the Control array, writing the extracted data into the Output array one element for each step.

If Bit 2 of the current Control array element is 0, MBDRV will copy the next byte of the source frame into the next Output array element. Note that bytes are treated as unsigned 8-bit integers, so the values returned for Byte elements range from 0 to 255. If Bit 2 is 1, MBDRV will transfer a Word by swapping and copying the next two bytes of the source frame into the next element of the Output array.

Bit 3 marks the end of the Control list. If any source frame bytes are left, they will be copied using the last valid Control word before the End marker. This feature is useful for commands like RDIS and RDOR, where you will want to copy all data after the header information as words, but you may not know exactly how many bytes are actually going to be in the frame until runtime. In this case, just make sure that the last two elements of the control array are 01xx and 10xx binary respectively.

> **Note:** BRKRCV examines only Bits 2 and 3 of the Control array elements. This can be important, because it allows you to place the SNDUSR and BRKRCV control masks into the same array for user-defined commands. BRKRCV never alters the contents of the Control array.

IMPORTANT! When using the BRKRCV command, it is *crucial* that the Control, Frame, and Output Arrays be in the same 80x86 data segment. This is normally not a problem unless you are using a language like Quick Basic or Turbo Pascal, languages that often place data in multiple data segments.

If you are using a language that does not allow you to control segment allocation, you should take precautions. For example, you could copy the Control, Frame, and Output array contents into separate vectors of a three-dimensional array (such arrays are always kept in a single segment). In Turbo Pascal, you can generally force the same segment by making sure that all three arrays are either local or global variables.

EXAMPLE

Suppose that you have used RCVUSR to retrieve the following incoming command string (in Hex): 16 10 00 87 00 02 04 00 0A 01 02

RCVUSR will return (in decimal) ADDR = 22, FC = 16 (Write Multiple Registers), RCVL = 11. Suppose also that the frame data is stored in an array called FRAR. The following code segment is designed to process the Write Multiple Registers command. In a practical application, this might be used to accept data written into the IBM PC by the MODBUS device.

```
DIM CTLAR(8),OUTAR(128)
CTLAR(0) = 0    ' Byte 0 : Address
CTLAR(1) = 0    ' Byte 1 : Function Code
CTLAR(2) = 4    ' Bytes 2,3 : Start Address
CTLAR(3) = 4    ' Bytes 4,5 : Quantity
CTLAR(4) = 0    ' Byte 6 : Byte Count
CTLAR(5) = 4    ' Bytes 7,8 : First Data
CTLAR(6) = 8    ' End of string
CALL MBDRV(MB.BRKRCV, STATUS, RCVL, CTLAR(0), FRAR(0),
OUTAR(0))
```

After the BRKRCV command returns, OUTAR will contain:

Element	Contents (Decimal)	
0	22	Destination Address
1	16	Function Code (Write Multiple Registers)
2	135	Starting Address (Equivalent to Holding Register 40136)

3	2	Number of Values
4	4	Byte Count
5	10	Data Value 1 (for 40136)
6	258	Data Value 2 (for 40137)

When BRKRCV hits element 6 of the Control array (the End flag), it will interpret all remaining bytes of the incoming frame according to the last Control word. In this case, the last Control word means "Move Word", so all of the remaining data bytes, regardless of how many there are, will be copied as words into the Output array starting at element 6.

Note that this example implies "Slave Mode" operation. In other words, the MODBUS device issues the Write Multiple Registers command *to* the IBM PC and awaits a reply. Therefore, it is **very important** that you transmit the proper acknowledgement string using SNDUSR. Otherwise, the MODBUS device will "time out", and may either report an error condition or begin retrying the operation.

SRBCONV --- Convert String to Register and Bit

SYNOPSIS

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

FNO = 15	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Input String	String
ARG2	Return Register	Unsigned Integer
ARG3	Return Bit	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.

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 (Octal) and that ARG3 is less than or equal to 17 (Octal).

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.

This function is included mainly for compatibility with the Reliance Electric AutoMate™ system, which makes extensive use of octal addressing.

EXAMPLE

```
REG = 0
BIT = 0
INPUT"Enter point number";I$
CALL MBDRV(MB.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 MBDRV(FNO, STATUS, ARG1, ARG2, ARG3, ARG4)

FNO = 16	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Input String	String
ARG2	Return Register	Unsigned Integer
ARG3		
ARG4		

DESCRIPTION

Converts an octal string to a binary register number returned in ARG2. If the string contains illegal characters, or if ARG2 is greater than 37777 (Octal), STATUS will be returned as nonzero.

This function is included mainly for compatibility with the Reliance Electric AutoMate™ system, which makes extensive use of octal addressing.

EXAMPLE

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

WORDAR --- Break Word into Array

SYNOPSIS

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

FNO = 17	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Input Value	Integer
ARG2	Output 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 = 40221
REGCT = 1
REGVAL = 0
DIM BAR(16)
CALL MBDRV(MB.RDOR, STATUS, REG, REGCT, REGVAL, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
CALL MBDRV(MB.WORDAR, STATUS, REGVAL, BAR(0), B, B)
PRINT"Register 40221's value in binary: ";
FOR LA=15 TO 0 STEP -1
    IF BAR(LA) THEN PRINT"1"; ELSE PRINT"0";
NEXT LA:PRINT
```

ARWORD --- Assemble Array into Word

SYNOPSIS

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

FNO = 18	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Input Array	Integer Array
ARG2	Output Value	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.

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 = 40221
REGCT = 1
REGVAL = 0
CALL MBDRV(MB.ARWORD, STATUS, BAR(0), REGVAL, B, B)
CALL MBDRV(MB.WRSR, STATUS, REG, REGVAL, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Register 40221 set to"REGVAL
```

STEXT --- Send Text to Port

SYNOPSIS

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

FNO = 19	Function Number	Integer
STATUS	Error Return	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.

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 -3 if the string could not be transmitted for some reason.

EXAMPLE

```
STR$ = "AT D 1 800 555 1212" + CHR$(13) + CHR$(10)
CALL MBDRV(B.STEXT, STATUS, STR$, B, B, B)
IF STATUS<>0 THEN PRINT"Communication Error "STATUS:STOP
PRINT"Dialing..."
```

IOMAP --- Control I/O Mapping

SYNOPSIS

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

FNO = 20	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Flag	Integer
ARG2		
ARG3		
ARG4		

DESCRIPTION

Normally, MBDRV "maps" the Register and Coil addresses that you pass to conform to the Protocol's specifications. For example, if you refer to Holding Register 40127, the actual binary address transmitted by MBDRV will be 136, as defined by the Protocol.

However, if you are not working with Gould equipment, or if you need to control the actual transmitted addresses, you can disable address mapping with this command.

Address Mapping is enabled by default. To disable Address Mapping, call IOMAP with ARG1 equal to 0. Any nonzero value enables mapping.

EXAMPLE

```
FLG = 0
CALL MBDRV(MB.IOMAP, STATUS, FLG, B, B, B)
PRINT"Address Mapping Disabled."
```

OPPAR --- Other Port Parameters

SYNOPSIS

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

FNO = 21	Function Number	Integer
STATUS	Error Return	Integer
ARG1	Parity	Integer
ARG2	Stops	Integer
ARG3	Data size	Integer
ARG4		

DESCRIPTION

This function is used to set serial port parameters not covered by the CHNPOR command.

ARG1 can be 0 = No Parity, 1 = Odd Parity, or 2 = Even Parity.

ARG2 can be 1 = 1 Stop Bit, 2 = 2 Stop Bits.

ARG3 can be 7 = 7 Data Bits or 8 = 8 Data Bits.

Since the MODBUS protocol operates in pure binary, you should normally use 8 data bits. The other parameters will be determined by your target device.

EXAMPLE

```
PAR = 2    ' Even Parity
STP = 1    ' 1 Stop Bit
DAT = 8    ' 8 Data Bits
CALL MBDRV(MB.OPPAR, STATUS, PAR, STP, DAT, B)
IF STATUS<>0 THEN
PRINT"Illegal Communications Setup "STATUS:STOP
```

KEYPORT --- Set Hardware Key port

SYNOPSIS

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

FNO = 22	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.

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
```