Allen-Bradley
Standard Driver Software
(Cat. No. 6001-F2E)

User's Manual

Preface 6-1
This Manual's Purpose 6-1
Audience 6-1
Related Publications 6-1
Product Overview 1-1
Why Use the Standard Driver Software? 1-1
Compatible Hardware and Software 1-1
Overview of the Function Calls 1-2
Software Considerations 1-3
Installing the Standard Driver Software 2-1
Chapter Objectives 2-1
What the Package Includes 2-1
What's on the Diskette 2-2
Installing the Standard Driver Hardware and Software 2-3
Planning Your Application Program 3-1
Chapter Objectives 3-1
Include Files 3-1
Overview of the Function Calls 3-2
Programming Considerations 3-2
Using the Function Calls 4-1
Chapter Objectives 4-1
Using Open_StdDrv() 4-1
Communicating on DH-485 with the 6001-F2E Standard Driver 4-2
Using Appl_StdDrv() 4-2
Using Send_StdDrv() 4-4
Using Get_ErrMsg() 4-6
Using Close_StdDrv() 4-7

Compiling, Linking, and Configuring Your Application Program . . . 5-1
Chapter Objectives 5-1
Compiling and Linking Your Application Program 5-1
Configuring Your Application Program 5-2
Application Program Examples A-1
Sample Program A-1
PLC-2 Unprotected Read A-4
PLC-2 Unprotected Write A-6
Diagnostic Loop Back A-8
Diagnostic Counter Read A-10
Diagnostic Status A-12
Diagnostic Counter Reset A-14
Specifying Message Packet Commands with Send_StdDrv() B-1
Chapter Objectives B-1
Formatting the Message Packet B-1
Message Packet Fields B-4
Supported Command Set B-6
Diagnostic Counters Reset B-7
Diagnostic Loop B-7
Diagnostic Read B-8
Diagnostic Status B-8
Unprotected Read B-9

Unprotected Write B-9
Protected Typed Logical Read with Three Address Fields (File, Element, Sub-element) B-10
Protected Typed Logical Write with Three Address fields (File, Element, Sub-element) B-12
Diagnostic Replies C-1
Diagnostic Status Reply C-1
Diagnostic Read Reply (Diagnostic Counters) C-4
Error Codes D-1
Error Codes D-1

Preface

This Manual's Purpose

This manual shows you how to:

- Install the 6001-F2E Standard Driver software
- Communicate with DH-485 stations via the 6001-F2E Standard Driver software

Audience

Use this manual and the 6001-F2E Standard Driver software if any of your application programs require information from devices such as the SLC-500 programmable controller. We assume you are familiar with the DOS operating system and C programming language.

Related Publications

See the following publications for information about communicating on the DH-485 network.

Refer to this manual	For
PC DH-485 Interface Module Installation Data (publication 1784-2.23)	installing the 1784-KR module
Data Highway/Data Highway Plus Protocol and Command Set User's Manual (publication 1770-6.5.16)	information on PLC commands and protocol
SLC-500 Advanced Programming Software User's Manual (publication 1747-801)	information on the SLC-500

Chapter 1

Product Overview

Why Use the Standard Driver Software?

The Standard Driver Software (cat. no. 6001-F2E) for the 1784-KR Interface Module lets you communicate directly to SLC-500 programmable controllers and other devices on the DH-485 network. You use a standard set of function calls to communicate with DH-485 stations. These function calls let you define your own message packets or request pre-defined message packets included in the software. See page 4-2 for more information about the message packets.

Compatible Hardware and Software

You can use the following hardware devices with the 6001-F2E Standard Driver software:

- 1784-T35 Plant Floor Terminal
- 1784-T50 Industrial Terminal
- 6120, 6121, and 6122 Industrial Support Computers
- IBM PC/XT and PC/AT
- Compaq Deskpro 286
- Compaq Portable II and III

If your computer is not listed above, consult your local Allen-Bradley sales office for compatibility information.

You can use the following with the 6001-F2E Standard Driver software:

If you use this programming device	Use this operating system
1784-T50 programming terminal[1]	Allen-Bradley DOS version 3.21
IBM PC/XT, PC/AT, or IBM-compatible programming terminal	DOS version 3.0 or later (Use the DOS version included with the programming terminal)

[1]If you are using a 1784-T50 that has DOS version 2.11 or earlier, you need to purchase the current version of Allen-Bradley DOS

Overview of the Function Calls

The 6001-F2E Standard Driver Software consists of a library of C programming language function calls that let your computer communicate with nodes on the DH-485 network. Table 1.A lists these commands:

Table 1.A
6001-F2E Standard Driver Software Commands

If you want to	Use this function call
perform initialization functions (required before any communication can take place)	Open_StdDrv
transmit data over the DH-485 network to a DH-485 station	Send_StdDrv
release all resources and services before the application program terminates	Close_StdDrv

See chapter 4 for more information on the function calls.

Software Considerations

Important information about the 6001-F2E Standard Driver software for the 1784-KR Interface Module is listed below:

- the 6001-F2E driver does not support unsolicited messages
- the 6001-F2E does not support multiple outstanding commands (you must receive the reply from your command before sending another command)
- the 6001-F2E supports the large memory model version of Microsoft C and Borland Turbo C only
- the 6001-F2E driver does not respond to a diagnostic status command sent to itself, but responds to diagnostic status commands sent from another computer
- the 6001-F2E driver does not return a local error code if the KR detects a node address on the network the same as its own. The station that had the address *first* stays on-line; the other station goes off-line, so check for duplicate addresses in this situation,
- if you incorrectly format messages to be sent by the Standard Driver software, a timeout condition occurs in the application. Since there is no error code indicating an incorrect message format, check the message format first when the application software times out with a timeout error code
- the 6001-F2E does not support off-link messages (messages sent across a bridge to another network)
- the 6001-F2E supports applications that communicate with a single 1784-KR only

Chapter 2

Installing the Standard Driver Software

Chapter Objectives

In this chapter, you learn about:

- what the Standard Driver package includes
- the contents of the Standard Driver diskette
- how to install the Standard Driver Software

What the Package Includes

You have one of these two packages:

Cat. No.	Product Description
6001–F2E	Stand-alone Standard Driver software for the 1784-KR
1747-F2E	Standard Driver software bundled with the 1784-KR hardware

Each package contains:

- one User's Manual (publication 6001-6.5.5)
- one 5-1/4" diskette and one 3-1/2" diskette (use the appropriate diskette for your system)
- software license

If you ordered the 1747-F2E, you received a 1784-KR board in addition to the items listed above.

What's on the Diskette

The Standard Driver diskette contains the following types of files:

- linkable large memory model standard driver library files
- application library files
- example application files

Standard Driver Library Files

Use the following files to build a linkable large memory model application using the Standard Driver.

This file	Contains
L_MSKR.LIB	a large memory model Microsoft v5.1 compatible 6001-F2E Standard Driver library module
L_TCKR.LIB	a large memory model Borland v2.01 compatible 6001-F2E Standard Driver library module
STDDRV.H	definitions and declarations required to compile a 6001-F2E Standard Driver application
KRDEFS.H	
START485.EXE	the 6001-F2E Standard Driver start-up and initialization program. Run the START485.EXE to initialize the 1784-KR before you run your application

Application Library Files

The following files contain application functions libraries that let you use pre-defined support routines (Application Libraries). Use these in applications using the basic command set in table 4.B on page 4-2.

This file	Contains
L_MSAPP.LIB	a large memory model Microsoft v5.1 compatible application library module
L_TCAPP.LIB	a large memory model Borland v2.0 compatible application library module

Example Files

The following files contain working 6001-F2E Standard Driver application examples:

This file	Contains
F2EDIAG.C	diagnostic routines
F2ESLC.C	unprotected read and unprotected write routines
SCREEN.H	definitions and declarations required to compile F2EDIAG.C or F2ESLC.C

Installing the Standard Driver Hardware and Software

This section tells you how to install the Standard Driver hardware and software.

Installing the Hardware

To install the Standard Driver hardware:

1. Set the 1784-KR memory address switches and jumper settings that are compatible with your computer system (See the 1784-KR Installation Data, publication 1784-2.23, for instructions on setting these switches.)

2. Record these addresses on a piece of paper. You will need them when you configure the software.

Installing the Software

To install the Standard Driver software:

1. Create a working directory in your computer's hard disk (C:\F2E, for example). Use this directory to build your application program(s).

2. Put the disk containing the Standard Driver files in disk drive A:> (We use drive A:> as a default.)

3. Copy *all* the files from the disk to that directory (**`Copy A:*.* C:\F2E`**).

This completes the installation procedure. The next chapter helps you plan your application program.

Chapter 3

Planning Your Application Program

Chapter Objectives

This section guides you through the process of planning an application program. It contains the following:

- include files you need to put in your program
- function calls
- programming considerations

Include Files

The include files contain declarations for the driver type you are using. Define them at the top of your application program. The 6001-F2E Standard Driver uses the following include files:

- KRDEFS.H
- STDDRV.H

Important: In addition to including header files in your application, you need to link with an appropriate Standard Driver and Application Library file (whether you are using the Application Library or not).

If you use multiple files, place KRDEFS.H and STDDRV.H in the main file. Only one source per executable can reference each include file. If additional source files within an executable reference the Standard Driver software, you must do the following:

- duplicate KRDEFS.H and STDDRV.H under different names
- delete the following lines from the duplicate KRDEFS.H:

```
int max_umsg = Max_Umsg;
int max__smsg = 16;
```

- delete from the duplicate STDDRV.H all lines beginning from the first occurance of `/* ..... */` under the “Prototype” heading to the end of the file

Use these edited versions in all remaining source files that reference the 6001-F2E Standard Driver software.

Overview of the Function Calls

Function calls let your application program communicate with devices on the DH-485 network:

This Function Call	Lets You
Open_StdDrv()	initialize the 6001-F2E Standard Driver. Use this function call in *every* program you write.
Appl_StdDrv()	use predefined support routines (Application Library) in applications that use the basic command set. See page 4-2 for a list of these commands.
Send_StdDrv()	format commands not provided in the Application Library. It lets you format message packets when communicating with token-passing or slave-only devices.
Get_ErrMsg()	retrieve an ASCII string that describes a network message error.
Close_StdDrv()	end communication. Use this function call in *every* program you write.

Important: When you write your program, you must *always* start with Open_StdDrv() and end with Close_StdDrv(). The middle of your program will consist of Appl_StdDrv() or Send_StdDrv() or a combination of both. Use the Send_StdDrv() function call to format commands not supported by the Application Library routines. (See table 4.B on page 4-2 for a list of these commands.)

Your disk contains example programs (F2EDIAG.C and F2ESLC.C) you can use to test communication on the DH-485 network. (See appendix A for additional program examples.)

The next chapter for the format and parameters for each function call.

Programming Considerations

Keep the following considerations in mind when you write your application program:

- Use Borland Turbo C (v2.01 or later) or Microsoft C (v5.0 or later)
- Whether you use only Appl_StdDrv() routines or Send_StdDrv() routines, you must *always* link the following:

If you are using this	Link these files
Borland Turbo C (v2.01)	L_TCKR.LIB
	L_TCAPP.LIB
Microsoft C (v5.0)	L_MSKR.LIB
	L_MSAPP.LIB

Chapter 4

Using the Function Calls

Chapter Objectives

This chapter shows you how to use each of the function calls. It includes the format and parameters for each function call.

Using Open_StdDrv()

The Open_StdDrv() function call initializes the 6001-F2E Standard Driver. To open the Standard Driver, use the following format and parameters:

Format for Open_StdDrv()

The Open function call is shown below:

```
status = Open_StdDrv(device,0,0,0,
                     (unsol_msg *)NULL,
                     0,0,0);
```

Parameters for Open_StdDrv()

Assign the parameters in Table 4.A:

Table 4.A
Assigning Parameters to Open Communication

Parameter	Type	Description
device[ ] = “KR:0”	char	Assigns a driver type “KR:” and communication channel “0”. The 6001-F2E Standard Driver supports one 1784-KR communication channel.
(unsol_msg *)NULL,	struct	This is a null pointer for this release.
0	N/A	These parameters are ignored, but you still need to include them in the function call. Type in a zero for each ignored parameter.

When the Open_StdDrv() function is called, a status value is returned indicating whether the operation was successful or unsuccessful. Normal completion is 1. A value other than 1 indicates that an error occurred.

See Appendix D for a list of error codes.

Communicating on DH-485 with the 6001-F2E Standard Driver

After you have initialized the Standard Driver with the Open_StdDrv() function call, you are ready to communicate. You can use the Appl_StdDrv() function call (Application Library) or the Send_StdDrv() function call or a combination of both. Use the SendStdDrv() function call to format commands not supported by the Application Library. See table 4.B for a list of commands supported by the Application Library.

Using Appl_StdDrv()

The Appl_StdDrv() function formats DH-485 messages for the basic command set and transmits them over the DH-485 network. Use the Appl_StdDrv() with the selected PLC application symbol and Appl_StdDrv() message block data structure. The application symbols and the message block data structure are defined in the STDDRV.H header files. Use the following format and parameters:

Format for Appl_StdDrv()

The Appl_StdDrv() function call is shown below:

```
status = Appl_StdDrv(SYMBOL, SD_FB *);
```

Parameter	Type	Description
SYMBOL	int	identifies the support routine symbol (see table 4.B)
SD_FB	struct	initializes the DH+ function block (see table 4.C)

See appendix D for return **`status`** values for the Appl_StdDrv() function.

Parameters for Appl_StdDrv()

Table 4.B below shows the available functions (in the Application Library) you can use with Appl_StdDrv(). Table 4.C shows the Appl_StdDrv() message block data structure:

Table 4.B
6001-F2E Support Routines

To do this	Specify this
Diagnostic loop back testing	PLC_DLB
Read diagnostic counters	PLC_DCR
Read diagnostic status	PLC_DS
Reset diagnostic counters	PLC_RC
Basic command set unprotected read	PLC_UPR
Basic command set unprotected write	PLC_UWR

Refer to Appendix B for each of the available Application Library routines.

Table 4.C
Appl_StdDrv() Message Block Data Structure

Variable	Type	Description
DHP_MSG.dev = device;	char	The device variable indicates to the Standard Driver the communication interface and its channel. Set this variable to KR:0.
DHP_MSG.stat = &io_stat[0];	unsigned int	The io_stat variable serves two purposes. When Application Library routines are called, they return status before any type of reply is received from the remote device. When the Appl_StdDrv() successfully initiates a request to send a message (status = 1), io_stat[0] is reset to 0. When a reply message is received or a reply timeout occurs that matches the original request, io_stat[0] is set to a value greater than 0. Normal completion is 1. If io_stat[0] is normal, io_stat[1] will contain the length of the reply data buffer. If io_stat does not equal 1, an error occurred. The format is as follows: The low byte (EXT STS) of io_stat[0] contains local errors, such as timeout. The High byte (STS) of io_stat[0] contains DH-485 errors. If the high byte of io_stat[0] equals F0 Hex (indicating the extended DH-485 status), the low byte if io_stat[0] will contain the extended DH-485 status value. See appendix D for more information on error messages. See table 4.F for examples of STS and EXT STS bytes in io_stat[0].
DHP_MSG.L_R = Loc_Rem;	int	Set this variable to 0.
DHP_MSG.dst = &destination;	unsigned char	The DH-485 destination address. The destination variable is the DH-485 address where you want your message to be sent on the DH-485 network.
DHP_MSG.dta = &dt_addr	unsigned int	The data table address. Depending on the routine, the dt_addr variable is a two byte value or string describing the data table address where data is to be read from or written to.
DHP_MSG.len = size;	int	When reading data, the size variable indicates how many data bytes are to be read. When writing data, the size variable indicates how many bytes should be copied from the data buffer and written to the remote station.
DHP_MSG.buf = &d_buff[0];	unsigned char or int	The application data buffer. The application data may take on several formats. If you are using byte values, d_buff may be defined as a char array, the size variable is defined as a one-to-one relationship. If you are using signed integers, data is automatically stored in byte swapped format, the size value is defined as a two-to-one relationship.
DHP_MSG.TO = timeout;	unsigned int	The timeout variable is the number of seconds that you want to wait for a reply.

Important: If you are communicating to an SLC-500 using the unprotected read or unprotected write commands, create Data File 9 in the SLC-500. The SLC-500 uses this file for DH-485 communication. The data table address (defined as dt_addr in the Appl_StdDrv() function) will

be interpretted by the SLC-500 as a logical offset (in words) into Data File 9. See the SLC-500 Advanced Programming Software Manual, chapter A3, for more information on the SLC-500 memory organization.

Using Send_StdDrv()

If you do not wish to use the commands supported by the Application Library, use the Send_StdDrv() function to send user-formatted messages. The Send_StdDrv() function call transmits data over the DH-485 network to a DH-485 station. Depending on your application, you can use the Send_StdDrv() function call two ways. You can:

- communicate with token-passing DH-485 devices (such as the SLC-500)
- communicate with slave-only DH-485 devices

(See appendix B for more information on message packet formats for the Send_StdDrv() function call.)

Any transmission that does not complete normally is aborted by a timeout or local error code indicating the problem. See appendix D for a list of error codes. Use the following format and parameters:

Format for Send_StdDrv()

The Send_StdDrv() function call is shown below:

```
status = Send_StdDrv(device,
                     &io_stat[0],
                     &cmd_buff[0],
                     pass_thru,
                     &reply_buff[0],
                     timeout,
                     0,0)
```

Use the Send_StdDrv() function call to send user-formatted messages.

Parameters for Send_StdDrv()

Assign the parameters in Table 4.D:

Table 4.D
Assigning Parameters to Send Data

Parameter	Type	Description
device[] = “KR:0”	char	The device parameter should coincide with the same device used in the Open_StdDrv() function (Set this value to KR:0.)
io_stat[2];	unsigned int	The io_stat variable serves two purposes. When standard driver routines are called, they return status before any type of reply is received from the remote device. When the Standard Driver successfully initiates a request to send a message (status = 1), io_stat[0] is reset to 0. When a reply message is received or a reply timeout occurs that matches the original request, io_stat[0] is set to a value greater than 0. Normal completion is 1. If io_stat does not equal 1, an error occurred. The format is as follows: The low byte (EXT STS) of io_stat[0] contains local errors, such as timeout. The high byte (STS) of io_stat[0] contains DH-485 errors. If the high byte of io_stat[0] equals F0 Hex (indicating the extended DH-485 status), the low byte if io_stat [0] will contain the extended DH-485 status value. See appendix D for more information on error messages. See table 4.F for examples of STS and EXT STS bytes in io_stat[0].
cmd_buff[...]	unsigned char	The cmd_buff parameter is the buffer containing your message to be sent to the remote station. Use the following format: LEN TYP DST SRC CMD STS TNS TNS DATA The LEN field contains the entire packet length, including LEN. The TYP field is the message type. Set this value to 0 or 5, depending on the message type. The DST field is the DH-485 destination where your message is sent. The SRC field is the local 1784-KR DH-485 address. This field can be set to 0. See appendix B for definitions of each field.
pass_thru = 1; pass_thru = 0;	int	The pass_thru parameter is a reply option for the Standard Driver. When pass_thru is set to 1, the entire reply message (the header and the data you requested) is placed in your buffer. When the pass_thru parameter is set to 0, only the data you requested (not the header) is placed in your buffer. The length is returned in io_stat[1].
reply_buff[...];	unsigned char	The reply_buff parameter tells the driver where to put a reply message to your application.The reply message is copied into your buffer using the following format: LEN TYP DST SRC CMD STS TNS TNS DATA
timeout = 5;	unsigned int	The timeout parameter is the number of seconds your application waits for a reply message.
0, 0	N/A	Parameters 7 and 8 are not used, but you still need to include them in the function call. Type in a zero for both parameters.

See Appendix D for return **`status`** values for the Send_StdDrv() function.

Preventing Reply Messages from Being Lost

To prevent reply messages from being lost, we provide a Get_tns() function. Use the following format:

```
x = Get_tns()
```

The Get_tns() function returns an unsigned integer value. Place this value in the two-byte TNS field of your DH-485 message prior to calling the Send_StdDrv() function. See page B-5 for more information on Get_tns().

Using Get_ErrMsg()

The Get_ErrMsg() function call supplies a message string for errors in data transmission. If an error occurs, call Get_ErrMsg() with the error code to get a message indicating the problem. Use the following format and parameters:

Format for Get_ErrMsg()

The Get_ErrMsg() function call is shown below:

```
Get_ErrMsg(err, ret_msg);
```

Parameters for Get_ErrMsg()

Table 4.E explains the parameters:

Table 4.E
Get_ErrMsg() Parameters

Parameter	Type	Description
err	unsigned int	is a copy of the error value returned in io_stat[0].
ret_msg	char	is a pointer to a character buffer at least 80 characters long. It contains the error message string that corresponds to the code in the err parameter. The error message is returned in this buffer.

Table 4.F explains how to read io_stat[0] to get the STS and EXT STS bytes.

Table 4.F
Reading io_stat[0]

If the STS and EXT STS bytes of io_stat[0] look like this	It is a
STS: 0 / EXT STS: 0	Command message. No error. The values in the EXT STS byte can be any values.
STS: F0 / EXT STS: 0B	Reply message with extended status information. Match the code in the low byte (EXT STS) to the code for the appropriate command in appendix D.
STS: X / EXT STS: Y	Reply message. X = status field of the DH-485 packet (this field contains remote errors returned by devices on the DH-485 network. See appendix D for a list of error codes. Y = user interface local error codes listed in appendix D.

Using Close_StdDrv()

The CloseStdDrv() function call releases all resources and services and must be called before the application terminates. Use the following format and parameters:

Format for Closing Communication

The Close_StdDrv() function call is shown below:

```
status = Close_StdDrv(device);
```

Parameters for Closing Communication

Assign the parameters in Table 4.G:

Table 4.G
Assigning Parameters to Close Communication

Parameter	Type	Description
device[ ] = “KR:0”;	char	The device parameter should coincide with the device used in the Open_StdDrv() function

When the Close function is called, a status value is returned indicating whether the operation was successful or unsuccessful. Normal completion is 1. A value other than 1 indicates that an error occurred (See appendix D for a list of error codes.)

Chapter 5

Compiling, Linking, and Configuring Your Application Program

Chapter Objectives

This chapter tells you what you need to do before your application program can communicate with other devices on the network. First, you compile and link the software, then configure it. See the instructions below.

Compiling and Linking Your Application Program

To compile and link the linkable driver application software, do the following:

1. Create your application program and "include" the KRDEFS.H and STDDRV.H files.

2. Compile your application program with the large memory model option.

3. Link your driver with an appropriate standard driver and application library module.

Important: In addition to including header files in your application, you need to link with an appropriate Standard Driver and Application Library file (whether you are using the Application Library or not).

4. Copy your application.EXE and START485.EXE programs to the directory you wish to use.

5. If the 1784-KR card switch settings are configured for other than default settings, enter an environment string. See the PC DH-485 Interface Module Installation Data (publication 1784-2.23) for more information.

The following are compile line examples of Microsoft C and Borland Turbo C:

- Microsoft C

```
C>CL /AL /Gs <PROGRAM.C> L_MSAPP.LIB L_MSKT.LIB
```

- Borland Turbo C

```
C>TCC -ml <PROGRAM> L_TCAPP.LIB L_TCKT.LIB
```

Configuring Your Application Program

To configure your application program:

1. Type **START485** followed by the communication option settings. For example, type:

   START485 a0 m31 cM b9600

   where:

This Variable	Is
a	the DH-485 address of the 1784-KR (valid addresses are 0-31)
m	the maximum node address on the Dh-485 link (valid addresses are 0-31)
c	the category of operation (this is always m)
b	the DH-485 baud rate (valid entries are 300, 1200, 2400, 4800, 9600, or 19,200 baud only)

Important: SLC-500 processors operate at 9600 or 19,200 baud only.

2. Press [ENTER].

This operation initializes the 1784-KR. Now that the 1784-KR board and driver are installed and configured, you can run your application program.

Appendix A

Application Program Examples

Sample Program

The following program is an application that shows how to write information to Data File 9 of an SLC-500 processor using the Application Library and the PLC-2 (Unprotected Write) function symbol. If you are communicating to an SLC-500 using the unprotected read or unprotected write commands, you must first create Data File 9 in the SLC-500. The SLC-500 uses this file for DH-485 communication. The data table address (defined as dt_addr in the Appl_StdDrv() function) will be interpretted by the SLC-500 as a logical offset (in words) into Data File 9. See the SLC-500 Advanced Programming Software Manual, chapter A3, for more information on the SLC-500 memory organization

```
/* ------------------------------------------------- */
/* A typical 6001-F2E Application                     */
/* Calling the Application Library via the APPL function dispatcher   */
/* Write 2 integers to a remote SLC-500                */

/* include <stdio.h>                                  */

/* 6001-F2E linkable driver include files:            */
#include “krdefs.h”
#include “stddrv.h”

/* Define the communication device                    */
char   device[] = “KR:0”

main()
              {
#define                 NORMAL 1    /* Normal F2E status    */
                        int  i
                        size,       /* Data qty in bytes    */
                        d_buff[4],  /* Integer data buffer  */
                        PLC_FCT;
char                    err_msg[80];
unsigned char  DST=2,               /* Byte wide destination */
unsigned int   io_stat[2],          /* Status variables      */
               fct_stat,
               /* dt_addr[2]={0,0},                         */
               timeout=5;           /* 5 second timeout      */
```

```
/* DH-485 function block description found in STDDRV.H         */
struct SD_FB  DHP_MSG;          /* DH-485 function block       */

          /* Start the 6001-F2E software                       */
          fct_stat = Open_StdDrv(device,0,0,0,
                              (struct unsol_msg *)NULL,0,0,0);

          /* If the open was unsuccessful display the error and exit  */
          if (fct_stat != NORMAL){
              Get_ErrMsg(fct_stat, err_msg);
              printf(”n%s\n”,err_msg);
              exit(1);
              }

/* Initialize the data to be sent to the SLC-500               */


d_buff[0] = 1;    /* Integer value 1 */
d_buff[1] = 2;    /* Integer value 2 */
size     = 4;     /* This example uses 2-byte integers so size is..  */
                  /* .... 2 times the number of integers sent        */

/* Initialize the DH-485 function block                        */
  DHP_MSG.dev  = device;      /* Communication device          */
  DHP_MSG.stat = &io_stat[0];  /* I/O status                   */
  DHP_MSG.L_R  = 0;
  DHP_MSG.dst  = &DST;        /* DH-485 address                */
  DHP_MSG.dta  = &dt_addr;    /* DT address                    */
  DHP_MSG.len  = size;        /* Size in bytes                 */
  DHP_MSG.buf  = &d_buff[0];  /* Write buffer                  */
  DHP_MSG.TO   = timeout;     /* Reply timeout in sec          */


PLC_FCT = PLC2_UWR;
                              /* Initialize a PLC function     */

/* Write 2 integer values to a SLC-500 file at remote station DST   */
fct_stat = Appl_StdDrv( PLC_FCT,&DHP_MSG);                          */
```

```
/* Fct_stat returns 1 if F2E snet the command successfully    */
if (fct_stat != NORMAL){
   Get_ErrMsg(fct_stat, err_msg);
   printf("%s/n",err_msg);
   fct_stat = close_StdDrv(device);
    exit(1);
    }

/* When a reply has been received or a timeout io_stat[0] will be set*/
while(!io_stat[0]);
printf("status reply received\n");
/* If io_stat[0] does not = 1 then an error occurred
if (io_stat[0] != NORMAL){
   Get_ErrMsg(io_stat[0],err_msg);
   printf("%s/n",err_msg);
    }

fct_stat = Close_StdDrv(device);
exit(1);
}
```

PLC-2 Unprotected Read

```
/* ------------------------------------------------- */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: PLC–2 Unprotected Read
SYMBOL:      PLC2_URD

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
            d_buff[...]
unsigned int  status,
            io_stat[2],
            dt_addr = 0110,
            timeout;

char        device[]= “KR:0”;
unsigned char destination;
struct SD_FB  DHP_MSG;
*/

  /* Initialize the DH-485 function block                          */
  DHP_MSG.dev  = device;               /* Communication device      */
  DHP_MSG.stat = &io_stat[0];           /* I/O status               */
  DHP_MSG.L_R  = 0;
  DHP_MSG.dst  = &destination;          /* DH-485 address           */
  DHP_MSG.dta  = &dt_addr;             /* DT address(byte location) */
  DHP_MSG.len  = size;                 /* Size in bytes (max 244)   */
  DHP_MSG.buf  = &d_buff[0];            /* Read buffer              */
  DHP_MSG.TO   = timeout;              /* Reply timeout in sec      */


PLC_FCT = PLC2_URD;      /* Initialize a PLC function */
```

```
  /* Call the Standard Driver Appl() function            */
  status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*
status = See 6001–F2E User’s Manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)

Binary address format: dt_addr[0] = lo_byte, dt_addr[1] = hi_byte

*/
```

PLC-2 Unprotected Write

```
/* ------------------------------------------------- */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: PLC–2 Unprotected Write
SYMBOL:      PLC2_UWR

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
            d_buff[...]
unsigned int  status,
            io_stat[2],
            dt_addr = 0110,
            timeout;
char     device[]= “KR:0”;
unsigned char destination;
struct SD_FB  DHP_MSG;
*/

  /* Initialize the DH-485 function block                              */
  DHP_MSG.dev  = device;                    /* Communication device    */
  DHP_MSG.stat = &io_stat[0];                /* I/O status             */
  DHP_MSG.L_R  = 0;
  DHP_MSG.dst  = &destination;              /* DH-485 address          */
  DHP_MSG.dta  = &dt_addr;                 /* DT address (byte location) */
  DHP_MSG.len  = size;                     /* Size in bytes (max 242)  */
  DHP_MSG.buf  = &d_buff[0];                /* Write buffer           */
  DHP_MSG.TO   = timeout;                  /* Reply timeout in sec    */

  PLC_FCT = PLC2_UWR;     /* Initialize a PLC function  */
```

```
  /* Call the Standard Driver Appl() function          */
status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*status = See 6001–F2E User’s manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)

Binary address format: dt_addr[0] = lo_byte, dt_addr[1] = hi_byte

*/
```

Diagnostic Loop Back

```
/* ---------------------------------------------------------------------- */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: Diagnostic Loop Back
SYMBOL:      PLCx_DLB

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
unsigned int    status,
            io_stat[2],
            timeout;
char          device[]= “KR:0”;
unsigned char destination;
unsigned char d_buff[...];
struct SD_FB  DHP_MSG;
*/

      /* Initialize the DH-485 function block                              */
      DHP_MSG.dev  = device;                  /* Communication device      */
      DHP_MSG.stat = &io_stat[0];             /* I/O status                */
      DHP_MSG.L_R  = 0;
      DHP_MSG.dst  = &destination;            /* DH-485 address            */
      DHP_MSG.dta  = (unsigned char *)NULL;
      DHP_MSG.len  = size;                    /* Size in bytes (max 243)
         */
      DHP_MSG.buf  = &d_buff[0];              /* Loop back buffer
      DHP_MSG.TO   = timeout;                 /* Reply timeout in sec      */

      PLC_FCT = PLCx_DLB;                     /* Initialize a PLC function */
```

```
/* Call the Standard Driver Appl() function                                    */
status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*
status = See 6001–F2E User’s manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)
*/
```

Diagnostic Counter Read

```
/* ------------------------------------------------------------------- */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: Diagnostic Counter Read
SYMBOL:      PLCx_DCR

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
unsigned int    status,
            io_stat[2],
            ctr_addr;
            timeout;
char          device[]= “KR:0”;
unsigned char destination;
unsigned char d_buff[...];
struct SD_FB  DHP_MSG;
*/

      /* Initialize the DH-485 function block                              */
      DHP_MSG.dev  = device;              /* Communication device          */
      DHP_MSG.stat = &io_stat[0];          /* I/O status                   */
      DHP_MSG.L_R  = 0;
      DHP_MSG.dst  = &destination;         /* DH-485 address               */
      DHP_MSG.dta  =  &ctr_addr;          /* Counter address ret’d from DS cmd */
      DHP_MSG.len  = size;                /* Size in bytes (max 243)       */
      DHP_MSG.buf  = &d_buff[0];           /* Counter buffer               */
      DHP_MSG.TO   = timeout;              /* Reply timeout in sec         */
```

```
      PLC_FCT = PLCx_DCR;                  /* Initialize a PLC function  */

/* Call the Standard Driver Appl() function                              */
status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*
status = See 6001–F2E User's manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)
*/
```

Diagnostic Status

```
/* ------------------------------------------------------------------ */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: Diagnostic Status
SYMBOL:      PLCx_DCR

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
unsigned int    status,
            io_stat[2],
            timeout;
char          device[]= “KR:0”;
unsigned char destination;
unsigned char d_buff[...];
struct SD_FB  DHP_MSG;
*/

      /* Initialize the DH-485 function block  */
DHP_MSG.dev  = device;                          /* Communication device  */
DHP_MSG.stat = &io_stat[0];                     /* I/O status            */
DHP_MSG.L_R  = 0;
DHP_MSG.dst  = &destination;                   /* DH-485 address        */
DHP_MSG.dta  = (unsigned char *)NULL;
DHP_MSG.len  = 0;                              /* Size not applicable   */
DHP_MSG.buf  = &d_buff[0];                     /* Status buffer         */
DHP_MSG.TO   = timeout;                        /* Reply timeout in sec  */
```

```
      PLC_FCT = PLCx_DS;            /* Initialize a PLC function       */

/* Call the Standard Driver Appl() function                            */
status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*
status = See 6001–F2E User’s manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)
*/
```

Diagnostic Counter Reset

```
/* ------------------------------------------------------------------ */
/*
DH/DH+ Application Library (c) 1989 Allen–Bradley

APPLICATION: Diagnostic Counter Reset
SYMBOL:      PLCx_RC

Include files:

Linkable Driver:
#include “krdefs.h”
#include “stddrv.h”

Application Variables:

int         size,
            PLC_FCT;
unsigned int  status,
            io_stat[2],
            timeout;
char          device[]= “KR:0”;
unsigned char destination;
unsigned char d_buff[...];
struct SD_FB  DHP_MSG;
*/

      /* Initialize the DH-485 function block                          */
      DHP_MSG.dev  = device;                    /* Communication device  */
      DHP_MSG.stat = &io_stat[0];                /* I/O status           */
      DHP_MSG.L_R  = 0;
      DHP_MSG.dst  = &destination;               /* DH-485 address       */
      DHP_MSG.dta  = (unsigned char *)NULL;
      DHP_MSG.len  = 0;                          /* Size not applicable  */
      DHP_MSG.buf  = (unsigned char *)NULL;
      DHP_MSG.TO   = timeout;                    /* Reply timeout in sec */
```

```
      PLC_FCT = PLCx_RC;              /* Initialize a PLC function    */

/* Call the Standard Driver Appl() function                          */
status = Appl_StdDrv( PLC_FCT, &DHP_MSG);
/*
status = See 6001–F2E User’s manual appendix D

io_stat[0]: Local & remote I/O status
            Hi_byte = DH-485 status
            Lo_byte = Local & extended DH-485 status
io_stat[1]:  Reply length in bytes (if applicable)
*/
```

Appendix B

Specifying Message Packet Commands with Send_StdDrv()

Chapter Objectives

Use the information in this appendix to format the Send_StdDrv() function call. Your message packet format depends on whether you are sending data to:

- token-passing devices (such as the SLC-500)
- slave-only DH-485 devices

This appendix contains:

- the message packet format you need (and an explanation of the fields each packet contains)
- supported commands for the Send function call

Formatting the Message Packet

The figures on the following pages show all possible fields in the Send function call message packet. The unshaded blocks indicate fields that are always included in your message packet. The shaded blocks indicate fields that may be included in your message packet, depending on the command. Use the appropriate message packet format for your particular application.

Communicating with a Token-passing Device (SLC-500 Processor)

Figure B.1 shows the message packet format you use to communicate with token-passing devices (such as an SLC-500 processor).

Important: Shaded blocks indicate packet fields that may or may not be included in your message packet, depending on the command.

Figure B.1
Message Packet Format for Communicating with an SLC-500 Processor

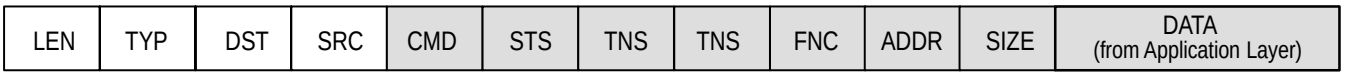

Table B.A explains the message packet fields in Figure B.1.

Table B.A
Communicating with an SLC-500 Processor

This Packet Field	Is the
LEN	length of the message packet (including LEN)
TYP	type code identifying this message packet format **(this is always a 0)**
DST	DH-485 destination address for this packet
SRC	DH-485 address of the device sending the packet
CMD	command byte
STS	message packet status
TNS	transaction number (2 bytes)
FNC	byte used with CMD. If used, it defines the specific function under the command type
ADDR	two-byte field that contains the address of a memory location in the command executor where the command is to begin executing.
SIZE	byte that specifies the number of data bytes to be transferred by a message
DATA	data sent as part of themessage command

Communicating with a Slave-only Device

Figure B.2 shows the message packet format you use to communicate with a slave-only DH-485 device.

Important: Shaded blocks indicate packet fields that may or may not be included in your message packet, depending on the command.

Figure B.2
Message Packet Format for Communicating with a Slave-only Device

Table B.B shows the message packet fields in Figure B.2.

Table B.B
Communicating with a Slave-only Device

This Packet Field	Is the
LEN	length of the message packet (including LEN)
TYP	type code identifying this format **(this is always a 5)**
IMMEDIATE BLOCK	immediate block in the destination device
1	local link message **(this is always a 1)**
DST	DH-485 destination address for this packet
SRC	DH-485 address of the device sending the packet
CMD	command byte
STS	message packet status
TNS	transaction number (2 bytes)
FNC	byte used with CMD. If used, it defines the specific function under the command type
ADDR	two-byte field that contains the address of a memory location in the command executor where the command is to begin executing.
SIZE	byte that specifies the number of data bytes to be transferred by a message
DATA	data sent as part of a message command

The next section describes the packet fields for both applications.

Message Packet Fields

The message packet fields for the previous applications are described in more detail in the table below.

Table B.C
Message Packet Field Description

Field	Description
LEN	This field identifies the length of the message packet (including LEN). The allowed range is 0-255.
TYP	This is the code that identifies the type of communication. Use 0 to communicate to an SLC-500 processor and 5 to communicate to a slave-only device.
IMMEDIATE BLOCK	The allowed range is 0 and 128-255 (depending on the device.)
DST and SRC	The DST (destination) byte contains the DH-485 node number of the node that is the ultimate destination of the message. The SRC (source) byte contains the DH-485 node number of the node that originated the message. The application layer supplies the DST value; the data link layer supplies the SRC value. Allowed values for these bytes are 0 to 31 (decimal).
CMD and FNC	The CMD (command) and FNC (functions) bytes (1 byte each) together define the activity to be performed by the command message at the destination node. Command defines the command type and FNC, if used, defines the specific function under that command type.
STS and EXT STS	The STS (status) and EXT STS (extended status) bytes indicate the status of the message transmission. In command messages, the application program should always set the STS value to 0. In reply messages, the STS byte may contain one of the status codes listed in appendix D . If the high byte = F0 Hex, there is extended status information in the EXT STS byte. Otherwise, there is no EXT STS byte. An STS value of 0 in a reply message means the command has been executed with no error. In a reply message, the STS byte is divided between application layer and link layer. The link layer uses bits 0 through 3 of the STS byte to report local errors (those errors that occur when the link layer attempts to transmit a message across the link). The application layer uses bits 4 through 7 of the STS byte (and in some cases, the EXT STS byte) to report remote errors (those errors that occur when the command executor at the destination node tries to execute the command message). Refer to appendix D for more information.

Field	Description
TNS	The TNS (transaction) bytes (2 bytes) contain a unique 16-bit transaction identifier. For each command message transmitted by your computer node, your application level software must assign a unique 16-bit transaction number. A simple way to generate this number is to use the Get_tns() function and store the value in the two TNS bytes of the new message. See page 4-6 for more information on Get_tns(). When the command initiator receives a reply to one of its command messages, it can use the TNS value to associate the reply message with its corresponding command. If the TNS value of a reply message matches the TNS value of a command message, then that reply is the appropriate one for that command. Note that the low byte (least significant bits) of your TNS value will be transmitted across the link before the high byte (most significant bits). At any instant, the combination of SRC, CMD, and TNS are sufficient to uniquely identify every message packet in transit. At least one of these fields in the current message must be different than the corresponding field in the last message received by a command executor. If none of these fields is different, the message is ignored. The process is called duplicate message detection.
ADDR	The ADDR (address) field (2 bytes) contains the address of a memory location in the command executor where the command is to begin executing. For example, if the command is to read data from the command executor, ADDR specifies the address of the first byte of data to be read. The first byte of the ADDR field contains the low (least significant) byte of the address, and the second byte contains the high byte of the address. The ADDR field specifies a byte address, not a word address, as in SLC programming.
SIZE	The SIZE byte specifies the number of data bytes to be transferred by a message. This field appears in read commands, where it specifies the number of data bytes that the responding node must return in its reply message. The allowed value for SIZE varies with the type of command.
DATA	The DATA field contains binary data from the application program. The number of data bytes in a message depends on the command or function being executed.

Supported Command Set

The supported command set includes commands that can generally be executed by any SLC processor. Use these commands with the Send_StdDrv() function call.

Command	CMD (Hex)	FNC (Hex)
SLC-500		
Protected Typed Logical Read with 3 address fields	0F	A2
Protected Typed Logical Write with 3 address fields	0F	AA
Basic		
Diagnostic Counters reset	06	07
Diagnostic Loop	06	00
Diagnostic Read	06	01
Unprotected Read	01	N/A
Unprotected Write	08	N/A
Diagnostic Status	06	03

Important: In the example formats that follow, CMD and FNC values are expressed in hexadecimal notation; all other values are in decimal form.

The message packet fields for each of the commands are described in more detail in the following pages. See page B-2 for more information on formatting message packets.

Diagnostic Counters Reset

This command resets to zero all the diagnostic timers and counters in the node interface module. The diagnostic status command gives the starting address for this block of counters and timers

Command Format

CMD 06	STS	TNS	FNC 07

Reply Format

CMD 46	STS	TNS

Diagnostic Loop

You can use this command to check the integrity of the transmissions over the communication link. The command message transmits up to 243 bytes of data to a node interface module. The receiving module should reply to this command by transmitting the same data back to the originating node.

Command Format

CMD 06	STS	TNS	FNC 00	DATA 243 bytes max.

Reply Format

CMD 46	STS	TNS	DATA 243 bytes max.

Diagnostic Read

This command reads up to 244 bytes of data from the PROM or RAM of the node interface module. You can use it to read the module's diagnostic timers and counters. When communicating to an SLC-500, set ADDR to 0 and SIZE to 10 (decimal). See appendix C for diagnostic read replies for DH-485 devices.

Command Format

CMD 06	STS	TNS	FNC 01	ADDR	SIZE

Reply Format

CMD 46	STS	TNS	DATA 244 bytes max.

Diagnostic Status

Your computer uses this command to read a block of status information from a DH-485 device. The reply contains the information in its DATA field. The status information varies with the type of device. See appendix C for diagnostic status replies for DH-485 devices.

Command Format

CMD 06	STS	TNS	FNC 03

Reply Format

CMD 46	STS	TNS	DATA 244 bytes max.

Unprotected Read

This command reads words of data from any area of PLC data table memory. Use the SIZE field to specify the number of bytes to be read. To specify a number of PLC words, SIZE should be an even value because PLC words are two bytes long. Data bytes are transferred low byte first. The address of a word should be even.

Command Format

CMD 01	STS	TNS	ADDR	SIZE

Reply Format

CMD 41	STS	TNS	DATA 244 bytes max.

Unprotected Write

This command writes words of data into any area of PLC data table memory.

Command Format

CMD 08	STS	TNS	ADDR	DATA 244 bytes max.

Reply Format

CMD 48	STS	TNS

Important: If you are communicating to an SLC-500 using the unprotected read or unprotected write commands, create a Data File 9 in the SLC-500. The SLC-500 uses this file for DH-485 communication. The data table address (defined as dt_addr in the Appl_StdDrv() function) will be defined as a logical offset (in words) into Data File 9. See the SLC-500 Advanced Programming Software Manual, chapter A3, for more information on the SLC-500 memory organization.

Protected Typed Logical Read with Three Address Fields (File, Element, Sub-element)

Command Format

Byte No.:	1	2	3,4	5	6	7	8	9	10
	CMD 0F	STS	TNS	FNC A2	BYTE SIZE	FILE NUMBER	FILE TYPE	ELEMENT NUMBER	SUB–ELMT. NUMBER

Table C.A gives a description of each field.

Table B.D
Protected Typed Logical Read with Three Address Fields

Byte	Description
CMD	0F
STS	Indicates the status of a message transmission
TNS	A unique 16-bit transaction number. Use it to match a reply message to its corresponding command, which contains the same number.
FNC	A2
BYTE SIZE	the size of data to be read (in bytes), not including the address fields or other overhead bytes
FILE NUMBER (or bytes 7a-7c)	Byte 7 addresses files 0-254 only. For higher addresses, byte 7a = FF expands FILE NUMBER to three bytes total. Use bytes 7b and 7c for expanded file address (low address byte first)
FILE TYPE	80H, 81H – Reserved 82H – Output 83H – Input 84H – Reserved 85H – Bit 86H – Timer 87H – Counter 88H – Control 89H – Integer
ELEMENT NUMBER (or bytes 9a-9c)	Byte 9 addresses elements 0-254. For higher addresses, byte 9A = FF expands ELEMENT NUMBER to three bytes total. Bytes 9b and 9c are used for expanded element address (low address byte first)
SUB-ELEMENT NUMBER (or bytes 10a-10c)	Byte 10 only addresses sub-elements 0-254. For higher addresses, byte 10a = FF expands SUB-ELEMENT NUMBER to three bytes total. Bytes 10b and 10c are used for expanded sub-element address (low byte first)

Reply Format

CMD 4F	STS	TNS	EXT STS (only if error)	DATA (244 bytes max)

The following are possible STS and EXT STS responses.

Table B.E
Possible Responses for Typed Logical Read with Three Address Fields

This Response	Means
STS	
00	success
10	command format incorrect
50	address problem
F0	extended STS
EXT STS	
1A	file already open

Protected Typed Logical Write with Three Address fields (File, Element, Sub-element)

Command Format

Byte No.:	1	2	3,4	5	6	7	8	9	10	
	CMD 0F	STS	TNS	FNC AA	BYTE SIZE	FILE NUMBER	FILE TYPE	ELEMENT NUMBER	SUB–ELMT. NUMBER	DATA 244 bytes max.

Table B.F
Protected Typed Logical Write with Three Address Fields

Byte	Description
CMD	0F
STS	Indicates the status of a message transmission
TNS	A unique 16-bit transaction number. Use it to match a reply message to its corresponding command, which contains the same number.
FNC	AA
BYTE SIZE	the size of data to be written (in bytes), not including the address fields or other overhead bytes
FILE NUMBER (or bytes 7a-7c)	Byte 7 addresses files 0-254 only. For higher addresses, byte 7a = FF expands FILE NUMBER to three bytes total. Bytes 7b and 7c are used for expanded file address (low address byte first)
FILE TYPE	80, 81 – Reserved 82 – Output 83 – Input 84 – Status 85 – Bit 86 – Timer 87 – Counter 88 – Control 89 – Integer
ELEMENT NUMBER (or bytes 9a-9c)	Byte 9 addresses elements 0-254 only. For higher addresses, byte 9a = FF expands ELEMENT NUMBER to three bytes total. Bytes 9b and 9c are used for expanded element address (low address byte first)
SUB-ELEMENT NUMBER (or bytes 10a-10c)	Byte 10 addresses sub-elements 0-254 only. For higher addresses, byte 10a = FF expands SUB-ELEMENT NUMBER to three bytes total. Bytes 10b and 10c are used for expanded sub-element address(low address byte first)
DATA	Low byte first

Reply Format

CMD 4F	STS	TNS	EXT STS (only if error)

Appendix C

Diagnostic Replies

Diagnostic Status Reply

The tables below show diagnostic status replies for the following:

- SLC-5/01 (Table C.A)
- APS Software (Table C.B)
- 1784-KR Interface Module (Table C.C)

Table C.A
Diagnostic Status Replies for the SLC-5/01

Byte	Description	Status Reply
1	Mode/Status	00 (Hex)
2	Type Extender	EE (Hex)
3	Extended Interface Type	1F(Hex)
4	Extended Processor Type – for rack type 1747-L51 – for 20-40 I/O	 18 1A
5	Series/Revision – Bits 0-4 – Bits 5-7	 0 (Release 1) 1 (Release 2), etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name 1747-LP11&LP14 1747-L20 1747-L30 1747-L40	(in ASCII) 5/01 500-20 500-30 500-40
Product Specific Information		
17	Major Error Code Word	(low byte)
18	Major Error Code Word	(high byte)

Byte	Description	Status Reply
19	Processor Mode Status/Control Word – Bit 0-4 mode: – Bit 5 – Bit 6 – Bit 7	 0 – Download 1 – Program 2 – Reserved 3 – Idle due to SUS instruction 4 – Reserved 5 – Reserved 6 – Run 7 – Test Continuous Scan 8 – Test Single Scan 9 – 31 Reserved Forces Active Forces Installed Communication Active
20	Processor Mode Status/Control Word (High Byte) Bit 0 Bit 2 Bit 5	 Protection Power Loss Load Memory Module on Mem Error Major Error – Process halted
21	Program ID	(low byte)
22	Program ID	(high byte)
23	Processor RAM size (in Kbytes)	
24	Bit 0 Bits 2-7:	Directory File Corrupted 00-1F (Program Owner Node Address) 3F (No Program Owner)

Table C.B
Diagnostic Status Reply for APS Software (with COM1 Port DH-485 connection)

Byte	Description	Status Reply (Hex)
1	Mode/Status Byte	00 (no modes)
2	Type Extender	EE
3	Extended Interface Type	20
4	Extended Processor Type	1B
5	Series/Revision: Bits 0-4 Bits 5-7	 0 (Release 1) 1 (Release 2). etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name= APS	(ASCII)
17-24	Product Specific Information	00

Table C.C
Diagnostic Status Reply for 1784-KR

Byte	Description	Status Reply (Hex)
1	Mode/Status Byte	00 (no modes)
2	Type Extender	FE
3	Extended Interface type	24
4	Not used	00
5	Series/Revision Bits 0-4: Bits 5-7:	 0 (Release 1) 1 (Release 2). etc. 0 (Series A) 1 (Series B), etc.
6-16	Bulletin Number/Name = “1784-KR”	ASCII
17-24	Product Specific Information	00

Diagnostic Read Reply (Diagnostic Counters)

The table below contains diagnostic read reply values for:

- SLC-500
- APS (COM1)
- 1784-KR Interface Module

Table C.D
Diagnostic Read Reply Values for SLC-500, APS, and 1784-KR

This Byte	Means
0	Total packets received, low byte
1	Total packets received, high byte
2	Total messagaes sent, low byte
3	Total messages sent, high byte
4	Retries
5	Retry limit exceeded
6	NAK, no memory sent
7	NAK, no memory received
8	Bad messages received
9	Reserved

Appendix D

Error Codes

Error Codes

This appendix contains local and remote error codes:

Table D.A
Local Error Codes

Error Codes (in Hex)	Message
1	Successful transmission
18	LLC Interface not initialized; SSAP already used or invalid SSAP
19	LLC Interface not initialized; KTLLC driver not installed
1A	Unsuccessful send request; SSAP not active or illegal
1B	Unsuccessful send request; Invalid packet size
1C	Unsuccessful send request; Host cannot access dual port
1D	Reply timeout; No reply data received
1E	Unsuccessful send request; Buffer not available
1F	DH-485 Interface not initialized
23	Bad function parameter
25	Invalid Channel
27	KR transmit timeout
28	Fatal solicited buffer memory allocation
29	Fatal timer buffer memory allocation
2A	Fatal timer buffer memory corrupted
2B	KR channel already open

Table D.B
Remote Error Codes

Error Codes (in Hex)	Message
STS	
00	Success
10	Command format incorrect
50	Address problem
60	Disallowed due to command protection
F0	Extended status
EXT STS	
0B	Access denied. Improper privilege
1A	File already open
1B	Procesor in edit mode. Not accessible

A

Application Library Files, 2-2
Application Program
- Compiling, Linking, and Configuring, 5-1
- Examples, A-1
  - diagnostic counter read, A-10
  - diagnostic counter reset, A-14
  - diagnostic loop back, A-8
  - diagnostic status, A-12
  - plc-2 unprotected read, A-4
  - plc-2 unprotected write, A-6
  - sample program, A-1
- Planning, 3-1

C

Command Set, Supported, B-6
- Diagnostic Counters Reset, B-7
- Diagnostic Loop, B-7
- Diagnostic Read, B-8
- Diagnostic Status, B-8
- protected typed logical with three address fields, B-10
- protected typed logical write with three address fields, B-12
- Unprotected Read, B-9
- Unprotected Write, B-9

Compatibility, Hardware and Software, 1-1

D

DH-485 Network, Communicating with the 6001-F2E on the, 4-2
Diagnostic Replies
- Diagnosstic Status Reply, C-1
- Diagnostic Status Reply
  - for APS software, C-3
  - for the SLC-5/01, C-1

Diagnostic Reply
- Diagnostic Read Reply (Diagnostic Counters), C-4
- Diagnostic Status Reply, for the 1784-KR, C-3

E

Error Codes, D-1
- Local, D-1
- Remote, D-2

Example Files, 2-3

F

Function Calls, 3-2
- Appl_StdDrv(), message block data structure, 4-3
- Appl_StdDrv(), using, 4-2
- Close_StdDrv(), using, 4-7
- Get_ErrMsg(), reading io_stat[0], 4-7
- Get_ErrMsg(), using, 4-6
- Open_StdDrv(), using, 4-1
- Overview, 1-2
- Send_StdDrv()
  - preventing reply messages from being lost, 4-6
  - specifying message packet commands, B-1
- Send_StdDrv(), using, 4-4

G

Get_tns() Function, 4-6

I

Include Files, 3-1
io_stat[0], 4-7

M

Message Packet Fields, B-4
Message Packets, Formatting, B-1
- communicating with slave-only devices, B-3
- communicating with Token-passing devices, B-2

P

Programming Considerations, 3-2

S

Standard Driver Library Files, 2-2
Standard Driver Software
- Considerations, 1-3
- Contents of, 2-1
  - application library files, 2-2
  - example files, 2-3
  - standard driver library files, 2-2
- Installing, 2-1
  - hardware, 2-3
  - software, 2-4
- Product Overview, 1-1
- Why you use it, 1-1

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

With offices in major cities worldwide

WORLD HEADQUARTERS
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414)382-2000
Telex: 43 11 016
FAX: (414)382-4444

EUROPE/INDIA/ MIDDLE EAST/AFRICA HEADQUARTERS
Allen-Bradley Europa B.V.
Amsterdamseweg 15
1422 AC Uithoorn
The Netherlands
Tel: (31)2975/60611
Telex: (844) 18042
FAX: (31)2975/60222

ASIA/PACIFIC HEADQUARTERS
Allen-Bradley (Hong Kong) Limited
2901 Great Eagle Center
23 Harbour Road
G.P.O. Box 9797
Wanchai, Hong Kong
Tel: (852)5/739391
Telex: (780) 64347
FAX: (852)5/834 5162

CANADA HEADQUARTERS
Allen-Bradley Canada Limited
135 Dundas Street
Cambridge, Ontario N1R 5X1
Canada
Tel: (519)623-1810
Telex: (069) 59317
FAX: (519)623-8930

LATIN AMERICA HEADQUARTERS
1201 South Second Street
Milwaukee, WI 53204 USA
Tel: (414)382-2000
Telex: 43 11 016
FAX: (414)382-2400

Publication 6001-6.5.5 – May 1990
Supercedes publication 6001-6.5.4 – December 1989

PRINTED IN USA
PN 404631101