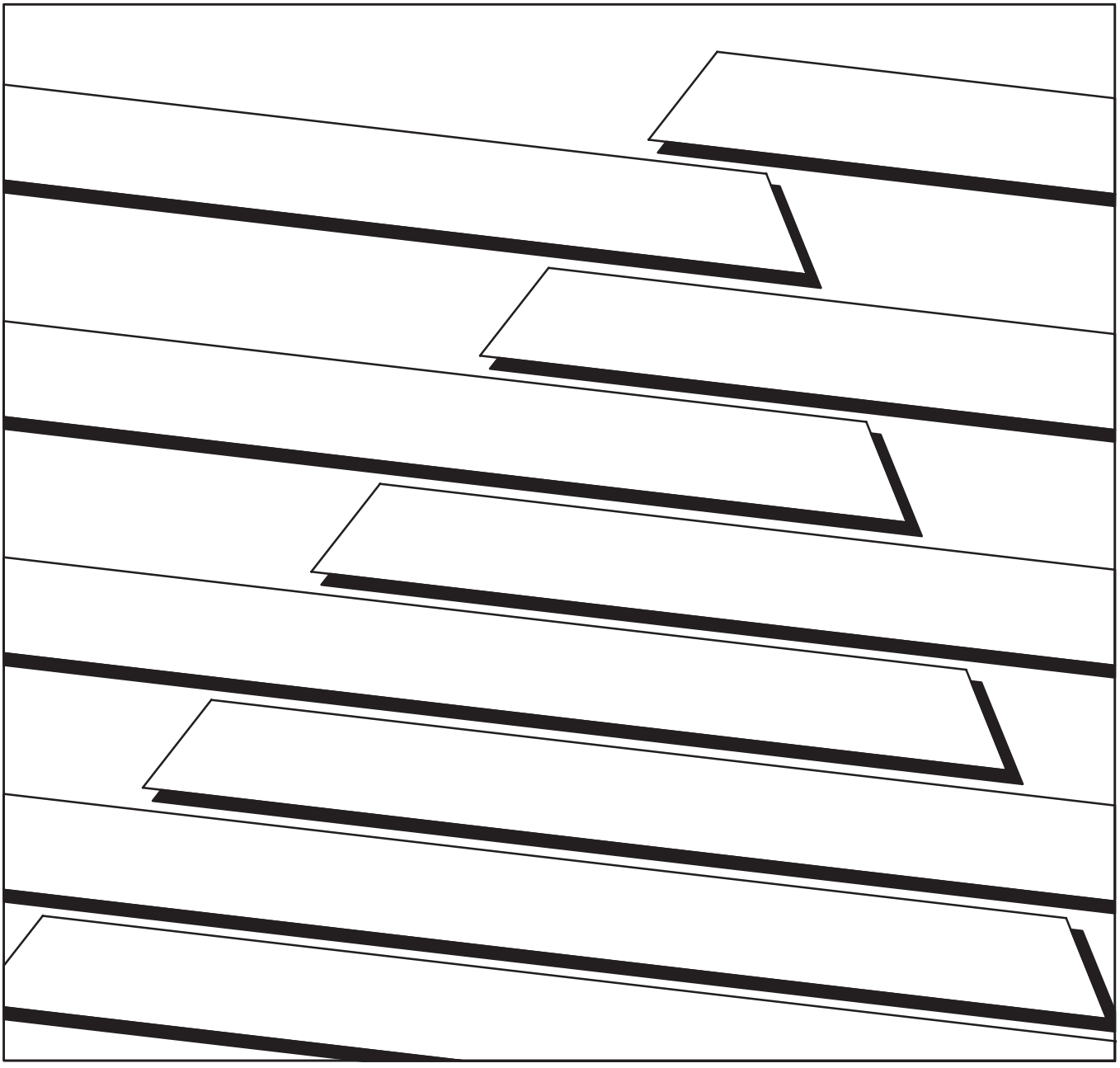
ALLEN-BRADLEY

1771 Control Coprocessor

(Cat. No. 1771-DMC, -DMC1, -DMC4, and -DXPS)

User Manual



Important User Information

Because of the variety of uses for the products described in this publication, those responsible for the application and use of this control equipment must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes, and standards.

The illustrations, charts, sample programs, and layout examples shown in this guide are intended solely for purposes of example. Since there are many variables and requirements associated with any particular installation, Allen-Bradley does not assume responsibility or liability (to include intellectual property liability) for actual use based on the examples shown in this publication.

Allen-Bradley publication SGI-1.1, Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control *(*available from your local Allen-Bradley office), describes some important differences between solid-state equipment and electromechanical devices that should be taken into consideration when applying products such as those described in this publication.

Throughout this manual, we use notes to make you aware of safety considerations:

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss.

Attention statements help you to:

- identify a hazard
- avoid the hazard
- recognize the consequences

Important: Identifies information that is critical for successful application and understanding of the product.

Summary of Changes

This edition of this publication contains new and updated information.

To help you find new and updated information in this manual, we have included change bars as shown to the left of this paragraph.

New Information

For detailed information on this subject:	See:
Three new function calls were added to the API library: • DTL_READ_W_IDX • DTL_RMW_W_IDX • DTL_WRITE_W_IDX	Appendix B
Two new Ethernet® communication features were added. You can now: • send/receive communication using Allen- Bradley's INTERCHANGE™ software and the INTERD daemon • use the SNMPD daemon	Chapter 6

Updated Information

For detailed information on this subject:	See:
All of the COMM ports on the coprocessor and expander are no longer initialized at the factory for connection to a terminal. The 9-pin serial port COMM 0, used for configuring the coprocessor, retains the factory settings for connection to a programming terminal. In Series A Revision E (1.30) and later of the firmware, however, COMM1, COMM2, and COMM3 have all of their serial-port settings prepared for raw binary data transfers.	Chapter 7
The control coprocessor and the expander now provide solid support for RS-485 communications. The modules now have the necessary hardware and low-level drivers on COMM1, COMM2, and COMM3.	

Using This Manual

Purpose of this Manual

Use this manual to help you install, configure, and operate your control coprocessor. This manual shows you examples of screens and programs to help you prepare your application programs.

Important: The programming-terminal screens and programs are examples only. Your applications may be different from the examples; therefore, the content of your screens and user programs may be different.

Referencing Other Control Coprocessor Documents

Table 1, Table 2, and Table 3 show the documents available with the control coprocessor.

Cat. No. 1771-PCB includes:

- PCBridge software disks, registration cards, and other software information contained in the Software Agreement Envelope
- Documentation Set D1771-L03

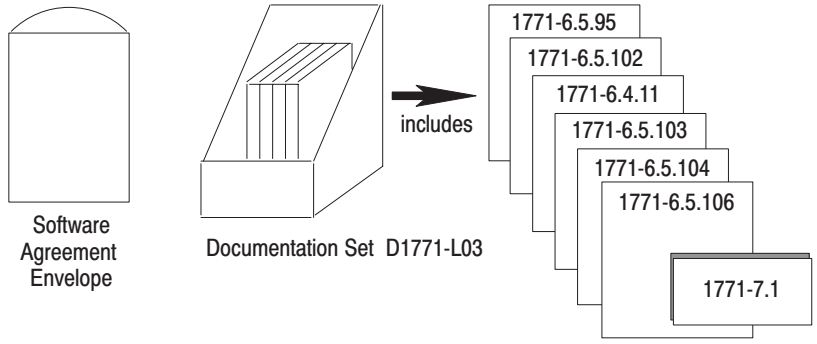
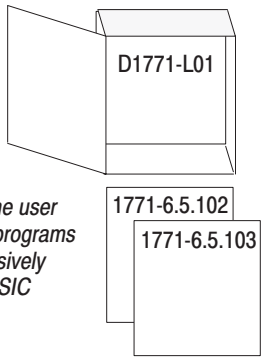


Table 1
PCBridge Documentation Set (D1771-L03)

Manual	Contents	Publication Number
1771 Control Coprocessor User Manual	Explains how to install, configure, and interface the control coprocessor to programmable controllers using the Allen-Bradley Interface Library	1771-6.5.95
OS-9 Operating System User Manual	Explains the OS-9 multi-tasking operating system and its utilities	1771-6.5.102
OS-9 Internet Software Reference Manual	Provides information on the TCP/IP protocol, FTP and TELNET utilities, and the socket library for client/server applications	1771-6.4.11
OS-9 BASIC User Manual	Shows BASIC program development	1771-6.5.103
OS-9 C Language User Manual	Provides information on C functions, the C compiler, and the source-code debugger	1771-6.5.104
OS-9 Assembler/Linker User Manual	Provides further information on programming in assembler and using the assembler language debugger	1771-6.5.106
1771 Control Coprocessor Error/Status Code Quick Reference	Contains a summary of error and status codes for the API library of functions, OS-9 operating system, compiler, assembler/linker, BASIC, and Internet	1771-7.1

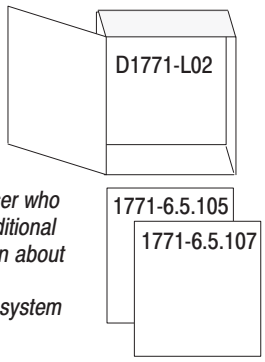


For the user who programs exclusively in BASIC

Table 2
BASIC Programming Reference (D1771-L01)

Manual	Contents	Publication Number
OS-9 Operating System User Manual	Explains the OS-9 multitasking operating system and its utilities	1771-6.5.102
OS-9 BASIC User Manual	Shows BASIC program development	1771-6.5.103

Table 3
OS-9 Technical Reference (D1771-L02)



For the user who needs additional information about the OS-9 operating system

Manual	Contents	Publication Number
OS-9 Technical I/O User Manual	Provides detailed information on writing device drivers	1771-6.5.105
OS-9 Technical Manual	Describes how memory modules are structured, loaded, linked, unlinked, etc.; describes how device drivers and device managers are structured, what functions they use to handle attached devices; also includes information on task scheduling, interprocess communication, pipes, interrupt processing, and alarms	1771-6.5.107

Going Through This Manual

Use the flow chart at the beginning of each chapter to determine where you are in the process of learning about the control coprocessor. To the left of the flow chart is a table that shows you the primary activities in the chapter and a page number for each activity.

Finding More Information

Contact your nearest Allen-Bradley office or distributor for more information about your control coprocessor or other Allen-Bradley products. For a list of publications with information about Allen-Bradley products, see the Allen-Bradley Publication Index, publication SD499.

Reporting Corrections and Suggestions

Use the Allen-Bradley Publication Problem Report, publication ICCG-5.21, to submit any corrections to or suggestions for this publication. Help us improve the quality of customer documentation.

Introducing the Control Coprocessor

Chapter 1

Chapter Objectives 1-1
Product Overview 1-1
Hardware Overview 1-3
Modes of Communication with a PLC Programmable Controller . 1-4
Programming Overview 1-6

Installing the Control Coprocessor

Chapter 2

Chapter Objectives 2-1
What You Need to Install Your Control Coprocessor 2-1
Select a Power Supply 2-2
Prevent Electrostatic Discharge Damage 2-2
Install the Control-Coprocessor Battery 2-3
Install the Keying Bands 2-5
Set Switch Configurations for the Main Module 2-6
Set Switch Configurations for the Serial Expander Module 2-6
Install the Control Coprocessor 2-7
Wire the Fault Relay 2-10
Apply Power to the Control Coprocessor 2-11
Remove the Control Coprocessor 2-11
What to Do Next 2-11

Getting Started with the Control Coprocessor

Chapter 3

Chapter Objectives 3-1
Connect the Programming Terminal 3-1
Select the Programming Interface 3-2
Install the Software on Your Personal Computer 3-2
Access the PCBridge Software 3-5
Configure Communication Parameters 3-6
Access and Use the OS-9 Command-Line Interface 3-7
Configure the Control Coprocessor 3-9
View Control-Coprocessor Current Status 3-19
Create a User Startup File 3-19
Send a Text File to the Control Coprocessor 3-20
Find Other OS-9 Commands 3-23
What to Do Next 3-23

Using the Programming Environment

Chapter 4

Chapter Objectives 4-1
Create a C Test Program 4-1
Compile a C Test Program 4-2
Send a Binary File to the Control Coprocessor 4-3
Confirm File Passage to the Control Coprocessor 4-5
Create a BASIC Test Program 4-5
Use an Example Application Program to Access the RAM Disk . 4-6
What to Do Next 4-8

Developing Programs

Chapter 5

Chapter Objectives 5-1
What Is the Application Program Interface 5-2
When to Use API Functions 5-2
How to Use DTL Functions 5-3
How to Use BPI Functions 5-6
How to Use Message Instructions 5-7
How to Use TAG Functions 5-10
How to Use CC Utility Functions 5-12
Prepare Programs for Direct-Connect Mode 5-14
Prepare Programs for Standalone Mode 5-18

Using the Ethernet Interface

Chapter 6

Chapter Objectives 6-1
Ethernet Communication 6-1
Connecting Ethernet to the Network 6-2
Addresses for the Ethernet Port 6-3
Modifying the Ethernet Configuration Files 6-4
Configuring the Ethernet Port 6-12
Using the OS-9/Internet FTP Utility 6-12
Using the OS-9/Internet TELNET Utility 6-17
Using the Internet Socket Library in C Programs 6-19
Using the INTERD INTERCHANGE Daemon 6-22
Using the SNMPD Daemon 6-27

Using the Serial Ports

Chapter 7

Chapter Objectives 7-1
Setting Up Communication Parameters 7-2
Referencing OS-9 Serial Port Device Names 7-4
Connecting to the Serial Port 7-4
Using a Serial Port for ASCII and Other Serial Communication . . 7-5
Using a Serial Port for RS-485 Communication 7-10
Using a Serial Port for RS-422 Communication 7-17

Interpreting Fault Codes and Displays

Chapter 8

Chapter Objectives 8-1
Serial Expander Module ASCII Display 8-1
Status for LEDs 8-2

Control Coprocessor Specifications

Appendix A

Product Specifications A-1
Product Compatibility A-2
Control-Coprocessor Memory A-2
CSA Certification A-3
UL Certification A-3

Application Program Interface Library of Functions

Appendix B

Appendix Objectives B-1
What Is the Application Program Interface B-1
Using Pointers B-2
BPI_DISCRETE B-3
BPI_READ B-5
BPI_WRITE B-8
CC_DISPLAY_DEC B-11
CC_DISPLAY_EHEX B-13
CC_DISPLAY_HEX B-15
CC_DISPLAY_STR B-17
CC_ERROR B-19
CC_ERRSTR B-21
CC_EXPANDED_STATUS B-23
CC_GET_DISPLAY_STR B-25
CC_INIT B-27
CC_PLC_BTR B-28
CC_PLC_BTW B-31
CC_PLC_STATUS B-34
CC_PLC_SYNC B-36
CC_STATUS B-38
DTL_C_DEFINE B-40
DTL_CLOCK B-43
DTL_DEF_AVAIL B-45
DTL_GET_FLT B-47
DTL_GET_WORD B-49
DTL_GET_3BCD B-51
DTL_GET_4BCD B-53
DTL_INIT B-55
DTL_PUT_FLT B-57
DTL_PUT_WORD B-59
DTL_PUT_3BCD B-61
DTL_PUT_4BCD B-63

DTL_READ_W B-65
DTL_READ_W_IDX B-67
DTL_RMW_W B-70
DTL_RMW_W_IDX B-73
DTL_SIZE B-76
DTL_TYPE B-78
DTL_UNDEF B-80
DTL_WRITE_W B-82
DTL_WRITE_W_IDX B-85
MSG_CLR_MASK B-88
MSG_READ_HANDLER B-90
MSG_READ_W_HANDLER B-94
MSG_SET_MASK B-98
MSG_TST_MASK B-100
MSG_WAIT B-102
MSG_WRITE_HANDLER B-105
MSG_WRITE_W_HANDLER B-109
MSG_ZERO_MASK B-113
TAG_DEF_AVAIL B-115
TAG_DEFINE B-116
TAG_GLOBAL_UNDEF B-119
TAG_LINK B-121
TAG_LOCK B-123
TAG_READ B-125
TAG_READ_W B-127
TAG_UNDEF B-129
TAG_UNLOCK B-131
TAG_WRITE B-133
TAG_WRITE_W B-135
Error Values B-137
BASIC Function Codes B-141

Cable Connections

Appendix C

Appendix Objectives C-1
Connecting to the 9-Pin COMM0 (/TERM) Port C-1
Connecting to the 25-Pin COMM1, 2, and 3 Ports C-3
Connecting to the Ethernet Port C-5

Using the PCBridge Software

Appendix D

Appendix Objectives D-1
About PCBridge Software D-1
Configuration Options D-1
Loading Memory Module D-4
Log On Remotely to OS-9 Terminal D-4
Modify Transfer List D-6
Modify Build List D-6
Using the Debugger D-7
Compiler Options D-9
Troubleshooting PCBridge Problems D-11
PCBridge Utilities D-11
binex/exbin D-12
cudo D-14
fixmod D-15
ident D-17
merge D-19
names D-20
os9cmp D-21
os9dump D-22

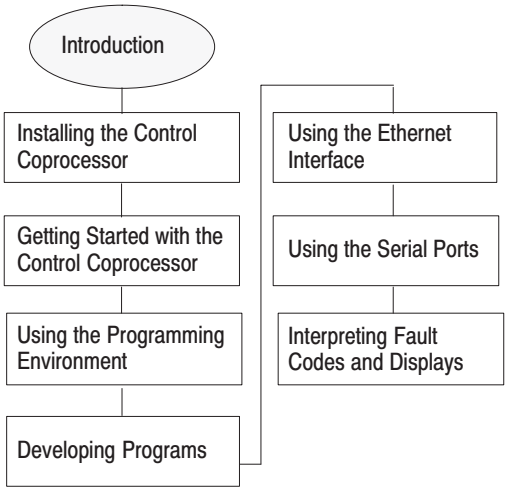
Chapter 1

Introducing the Control Coprocessor

Chapter Objectives

This chapter introduces the applications and functions of the control coprocessor. The chapter also covers the hardware components and the programming capabilities of a control coprocessor.

For:	See page:
Product overview	1-1
Hardware overview	1-3
Modes of communication with PLC programmable controllers	1-4
Programming overview	1-6



Product Overview

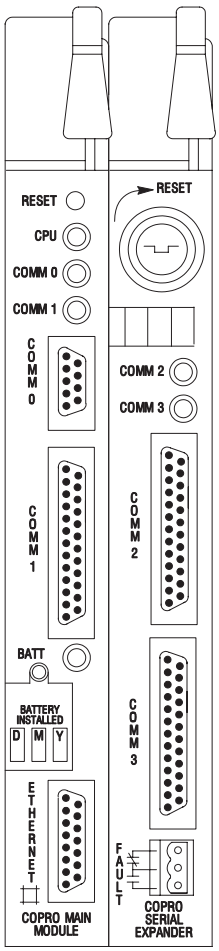
The control coprocessor expands the capability of a programmable-controller system by running C, BASIC, and assembler programs that perform tasks such as:

- manipulating and analyzing input, output, and other information gathered from the programmable controller
- communicating with devices external to the programmable controller system via the Ethernet® or asynchronous serial communication port(s)

These user programs run asynchronously to, and independently of, the programmable-controller control logic, but they do have access to its memory. You can use the control-logic programs in your programmable controller to start and stop your C, BASIC, or assembler programs.

You can use the control coprocessor for applications such as:

- calculating complex math or application-specific algorithms using C and/or BASIC programs
- production scheduling or historical-data logging/tracking
- high-speed search and compare of very large files or look-up tables
- protocol conversion for interfacing a programmable controller with a variety of field devices



Control-Coprocessor Modules

The control coprocessor consists of a main module and an optional serial expander module. Table 1.A lists catalog numbers for the control-coprocessor main modules and the optional serial expander module.

Table 1.A
Control-Coprocessor Catalog Numbers①

Control-Coprocessor Module Selection	Catalog Number
Main module—256 Kbytes	1771-DMC
Main module—1 Mbyte with Ethernet	1771-DMC1
Main module—4 Mbyte with Ethernet	1771-DMC4
Serial expander module	1771-DXPS

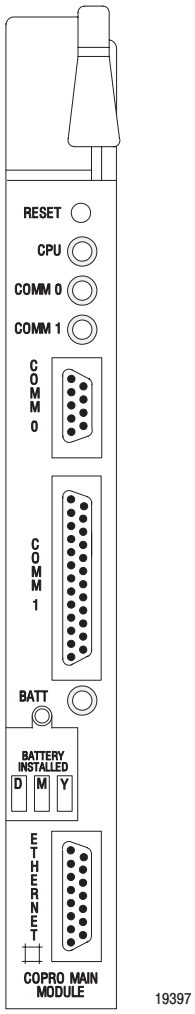
① See Appendix A, Table A.3 for more detailed information on memory usage.

The control-coprocessor main module is a 1-slot module. If you use both the control-coprocessor main module and a serial expander module, you require 2 slots. The figure at the left shows a 2-slot module with a 1771-DMC1 or -DMC4 main module and a serial expander module.

The control coprocessor is a member of the 1771 Universal I/O System. You can use it with or without a programmable controller.

Hardware Overview

Table 1.B describes the hardware elements for the main module.

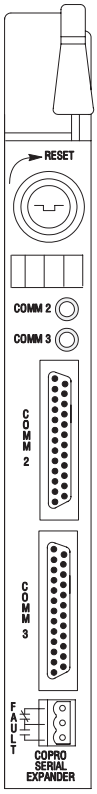


19397

Table 1.B
Main-Module Hardware Elements

Hardware Element	Description
RESET Switch	Use the reset switch to reinitialize the control coprocessor When the serial expander module is installed, use the keyswitch to reinitialize the coprocessor
LEDs	Four status indicators provide information on the CPU, COMM0 port, COMM1 port, and battery
COMM0 Port	This is a 9-pin, optically isolated, serial communication port that supports communication defined by EIA RS-232C standards Use this port to connect: • personal computers • terminals • other peripheral devices
COMM1 Port	This is a 25-pin, optically isolated, serial communication port that supports communication defined by EIA RS-232C, -423, and -485 standards You can also use this port with most RS-422A equipment as long as: • termination resistors are not used • the distance and transmission rate are reduced to 200 ft at 19.2 kbps Use this communication port to connect peripheral devices such as: • personal computers • terminals • bar-code readers • weigh scales • printers
Battery	This battery provides backup power for control coprocessor memory during power failure or normal down time Use the 3.0 volt lithium battery (cat. no. 1770-XYC) that is provided with your coprocessor
Ethernet Port (1771-DMC1 and 1771-DMC4 only)	The 1771-DMC1 and 1771-DMC4 versions of the control coprocessor include an Ethernet communication port that connects to thick-wire, thin-wire, or twisted-pair networks via a standard 15-pin transceiver connection These modules use TCP/IP protocol and have resident FTP and TELNET utilities You can program client/server applications for an Ethernet port using the TCP/IP socket library; an Internet socket library is supplied with the PCBridge software A downloadable driver is also available—as a part of the PCBridge software—that provides INTERCHANGE™ server functionality; when the coprocessor is attached to a standard PLC-5 processor, this provides Ethernet connectivity
Optional RAM	You can install additional RAM in the main module to expand user memory The following single inline memory modules (SIMMs) are available: **Memory Size** — **Catalog Number** 256 Kbytes — 1771-DRS 1 Mbyte — 1771-DRS1 4 Mbytes — 1771-DRS4

Table 1.C describes the hardware elements for the optional serial expander module.

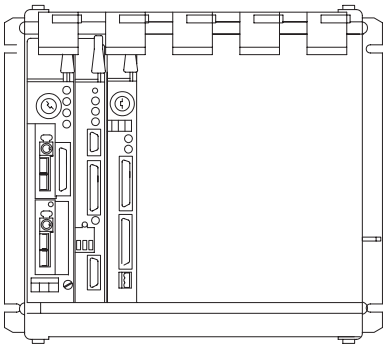


19397

Table 1.C
Serial Expander Module Hardware Elements

Hardware Element	Description
Keyswitch	This is a 2-position, spring-loaded keyswitch The RESET position is used to reinitialize the control coprocessor without cycling power
ASCII Display	The 4-character alphanumeric display shows information on the state of the control coprocessor, as provided by user programs
LEDs	The two status indicators provide information on the COMM2 and COMM3 ports
COMM2 Port and COMM3 Port	These are 25-pin, optically isolated, serial communication ports that support communication defined by EIA RS-232C, -423, and -485 standards You can also use the port with most RS-422A equipment as long as: • termination resistors are not used • the distance and transmission rate are reduced to 200 ft at 19.2 kbps Use these communication ports to connect peripheral devices such as: • terminals • personal computers • bar-code readers • weigh scales • printers
Fault Relay	The relay contact switches on a detected main-module hardware fault; the relay will handle 500 mA at 30 Vac/dc (resistive)

Modes of Communication with a PLC Programmable Controller


19398

A control coprocessor and serial expander in direct-connect communication with a PLC-5 programmable controller

The control coprocessor communicates with a programmable controller through a direct connection to the programmable controller—direct-connect mode—or via the 1771 I/O chassis backplane—standalone mode.

When you use the serial expander module in either mode, place it immediately adjacent to the main module under the same locking tab.

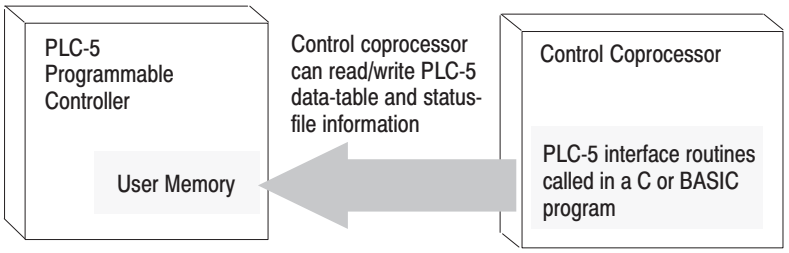
Direct-Connect Mode

In direct-connect mode, either the control coprocessor or the PLC-5® programmable controller initiates communications. The control coprocessor can read from and write to the PLC-5 programmable controller data table asynchronously to the ladder-program scan.

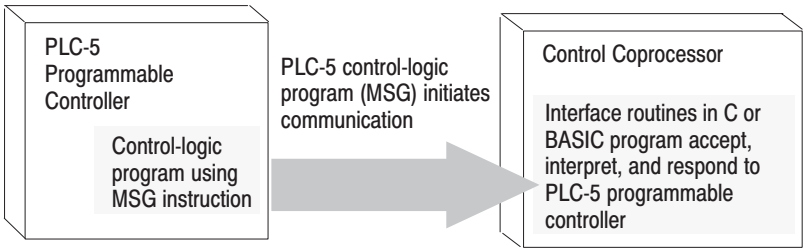
You can directly connect the control coprocessor to a PLC-5 programmable controller that has the coprocessor expansion port—e.g., a PLC-5/11™, PLC-5/20™, PLC-5/20E™, PLC-5/30™, PLC-5/40™ (series B, revision B or later), PLC-5/40E™, PLC-5/40L™, PLC-5/60™ (series B, revision B or later), PLC-5/60L™, PLC-5/80™, and PLC-5/80E™ programmable controller.

The **control coprocessor can initiate direct-access communication** to PLC-5 user memory as shown here.

You do not need to program your PLC-5 programmable controller to support these calls.



A **PLC-5 control-logic program can initiate direct-access communication** to the control processor as shown here.



A **PLC-5 control-logic program can initiate back-plane communication** with the control processor in direct-connect mode via:

- discrete I/O read/write
- block-transfer read/write

Important: If a fault occurs in a control coprocessor that is connected to certain PLC-5 programmable controllers via the side connector, the programmable controller may be unable to clear the major fault word without first resetting the control coprocessor. In extreme cases, you may need to separate the control coprocessor from the programmable controller. The programmable controllers on which this may be necessary are:

- PLC-5/60 (series B revision B)
- PLC-5/40 (series B revision B)
- PLC-5/30 (series A revision B)
- PLC-5/20 (series A revision A)
- PLC-5/11 (series A revision A)

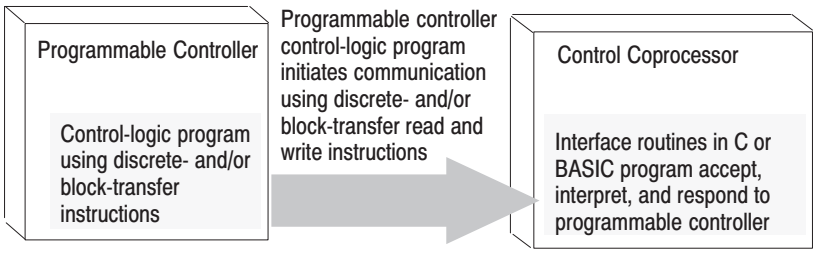
Standalone Mode

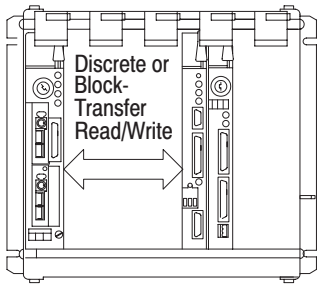
Tip We recommend that you use 1-slot addressing for standalone mode.

In standalone mode, you do not connect the control coprocessor directly to the PLC programmable controller. The control coprocessor can reside in the same chassis as the programmable controller or in a remote chassis.

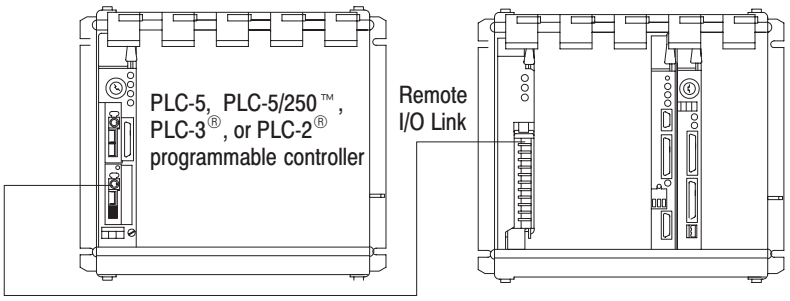
Only the programmable controller initiates communication with a standalone control coprocessor.

Communication is via the back-plane using discrete- or block-transfer read/write instructions.





Communicates, **in the same chassis**, with a PLC-5 or mini PLC-2® programmable controller via the backplane



Communicates, **from a remote chassis**, with the programmable controller via an I/O adapter module (1771-ASB)

19400

Programming Overview

This section provides an overview of the programming interface and capabilities of the control coprocessor.

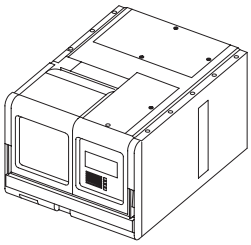
ATTENTION: Control-coprocessor programs that unintentionally write to memory outside their own data space can corrupt memory for other applications or corrupt system memory. We strongly recommend that development be done in an offline or non-critical context.

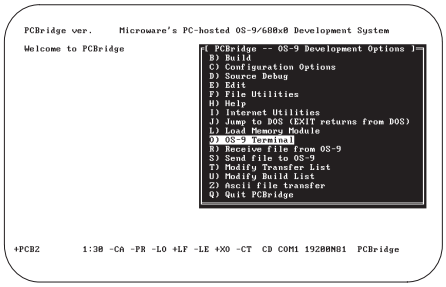
User Interface

You can develop programs and communicate with the control coprocessor using a DOS-based computer or an ASCII terminal. See Table 1.D.

Table 1.D
Programming Terminals

With this device:	You can:
DOS-based computer	• initialize and configure the control coprocessor • initialize and configure the Ethernet port • develop C, BASIC, and assembler programs • perform program debugging • initiate and terminate tasks using the OS-9 operating system command-line interface
ASCII terminal	• develop BASIC programs • perform program debugging • initiate and terminate tasks using the OS-9 operating system command-line interface





Program-Development Software

The PCBridge software package (1771-PCB) operates on a DOS- based personal computer. This software package supports offline and online user activities.

Use this software to:

- download/upload files and executable modules—or files and modules—to/from the control coprocessor
- develop and edit source files
- compile, assemble, and link multiple source files written in C or assembler
- emulate an ASCII terminal, which allows your personal computer to act as a console device to the control coprocessor
- use various online programs, such as basic (BASIC language environment) and SrcDbg (source-level debugger for C programs)
- access configuration (offline options) and other miscellaneous utilities
- initialize and configure the Ethernet port

Control-Coprocessor Operating System

The control-coprocessor operating system is Microware OS-9™. This real-time, multitasking operating system offers:

- command-line interface
- semaphore utilities
- inter-task communication facilities
- run-time task creation and deletion facilities
- task-prioritization facilities
- task-scheduling utilities
- unified I/O and file system for access to RAM disk and communication ports

See the OS-9 Operating System User Manual, publication 1771-6.5.102, for more information.

Programming Languages

You develop C, BASIC, and assembler programs using the PCBridge software. You can also develop and edit BASIC programs on the control coprocessor using a terminal or a personal computer for terminal emulation.

See the OS-9 C Language User Manual, publication 1771-6.5.104; the OS-9 Assembler User Manual, publication 1771-6.5.106; and the OS-9 BASIC User Manual, publication 1771-6.5.103, for more information on these languages.

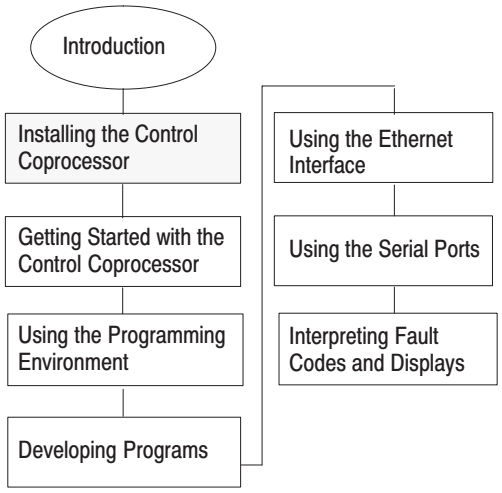
Chapter 2

Installing the Control Coprocessor

Chapter Objectives

This chapter provides instructions on how to install your control-coprocessor main module and serial expander module.

For information on:	See page:
What you need to install your control coprocessor	2-1
Selecting a power supply	2-2
Preventing ESD damage	2-2
Installing the battery	2-3
Installing the keying bands	2-5
Setting switch configurations	2-6
Installing the control coprocessor	2-7
Applying power to the control coprocessor	2-11
Removing the control coprocessor	2-11

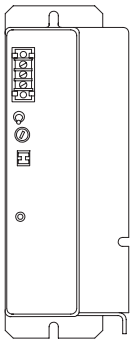


What You Need to Install Your Control Coprocessor

You need the following items for installation:

- control coprocessor
- serial expander module (optional)
- connector header (when using direct-connect mode)
- four connecting screws and spacers (when using direct-connect mode)
- lithium battery, battery cover, and mounting screw
- ESD grounding wrist strap
- chassis keying bands
- power supply
- chassis (properly grounded)

Select a Power Supply



1771-P7

Add current for:	
1. I/O modules	_____
2. PLC or adapter module	+_____
3. Main module	+_____
4. Ethernet (*)	+_____
5. Expander module (*)	+_____
6. Total current required	=_____

(*) See the associated step for how to determine if a value or what value should be added.

Before you install your control coprocessor, select an appropriate power supply. See the Control, Communication and Information Product Catalog, publication ICCG-1.1, for backplane current requirements. To determine the size of power supply that you require:

1. Record the total current draw for all I/O modules in the chassis.

2. Record the current draw for the programmable controller or adapter module in the chassis.

3. Record 2.50 Amps for each control-coprocessor main module in the chassis.

4. When you have a main module with Ethernet (1771-DMC1 or 1771-DMC4), record 3 times the current draw for your transceiver. If your transceiver requires 300 mA, for example, record 900 mA (or .90 Amps) as the result of:

```
300 mA x 3
```

5. When you use a serial expander module, record 1.50 Amps for each module in the chassis.

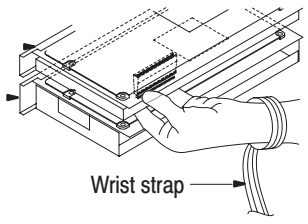
6. Total the values recorded in steps 1 through 5.

7. Select a power supply dependent on the input voltage required and total current requirements recorded in step 6.

8. Select a cable for the power supply.

Important: You cannot use an external power supply and a slot-based power supply to power the same chassis—they are not compatible.

Prevent Electrostatic Discharge Damage



The control coprocessor is shipped in a static-shielded container to guard against electrostatic electrostatic discharge (ESD). ESD can damage integrated circuits or semiconductors in the module if you touch backplane connector pins. ESD can also damage the module when you set configuration plugs or switches or add a SIMM (RAM memory). Avoid electrostatic damage by observing the following precautions:

- Remain in contact with an approved ground point while handling the module (by wearing a properly grounded wrist strap).
- Do not touch the backplane connector or connector pins.
- When not in use, keep the module in its static-shielded container.

Install the Control-Coprocessor Battery

The 1770-XYC battery ships with the control coprocessor and requires special handling. See Allen-Bradley Guidelines for Lithium Battery Handling and Disposal, publication AG-5.4.

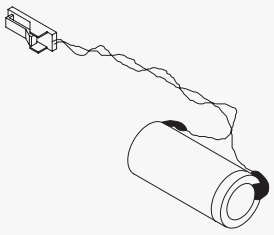
A red BATT status LED on the main module indicates that the battery needs replacement. Replace the battery while the module is powered so that your programs are maintained in memory. You may lose your programs if you remove the battery when power is removed.

ATTENTION: To maintain CSA certification for hazardous areas, do not substitute any other battery for the 1770-XYC.

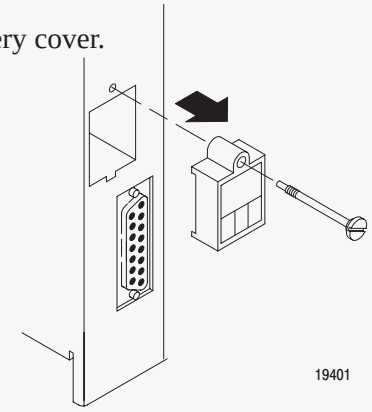
Installing the Control-Coprocessor Battery

You can install the battery either before or after you install the control coprocessor in the I/O chassis. To install the battery in the main module:

1. Remove the battery from the shipping bag.

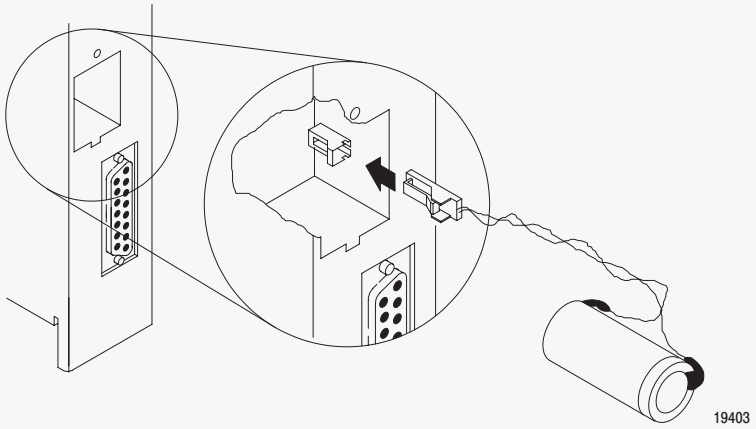


2. Remove the battery cover.



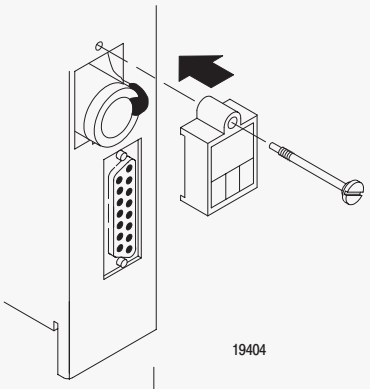
3. Remove any existing battery by pressing the lever on the battery-side connector and sliding the connectors apart.

4. Connect the battery.

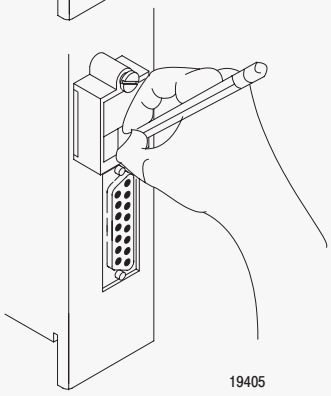


5. Place the battery and the wires in the main module.

6. Install the battery cover.



7. Using an erasable marker, record the battery-installation date.



Disposing of the Battery

Refer to the Allen-Bradley Guidelines for Lithium Battery Handling and Disposal, publication AG-5.4.

Do not dispose of lithium batteries in a general trash collection when their combined weight is greater than or equal to 1/2 gram. A single 1770-XYC battery contains .65 grams of lithium. Check your state and local regulations that deal with the disposal of lithium batteries. Follow these guidelines when you dispose of a control-coprocessor battery:

ATTENTION: Follow these precautions:

- Do not incinerate or expose the battery to high temperatures.
- Do not solder the battery or leads; the battery could explode.
- Do not open, puncture, or crush the battery. It could explode; and toxic, corrosive, and flammable chemicals could be exposed.
- Do not charge the battery. An explosion might result, or the cell might overheat and cause burns.
- Do not short positive or negative terminals together. The battery will heat up.

Install the Keying Bands

You receive plastic keying bands with each I/O chassis. Insert the keying bands in the backplane sockets of the I/O chassis, using the numbers beside the backplane connector as a guide. See Figure 2.1 and Figure 2.2.

Figure 2.1
Keying Band Positions for the Main Module

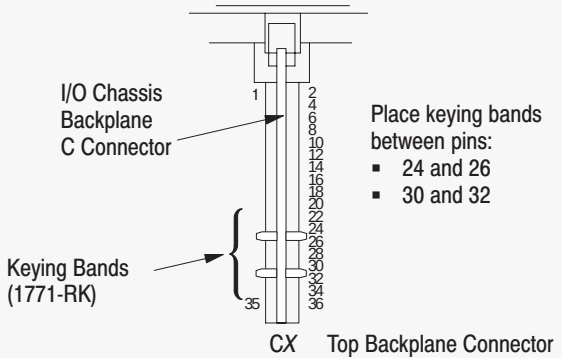
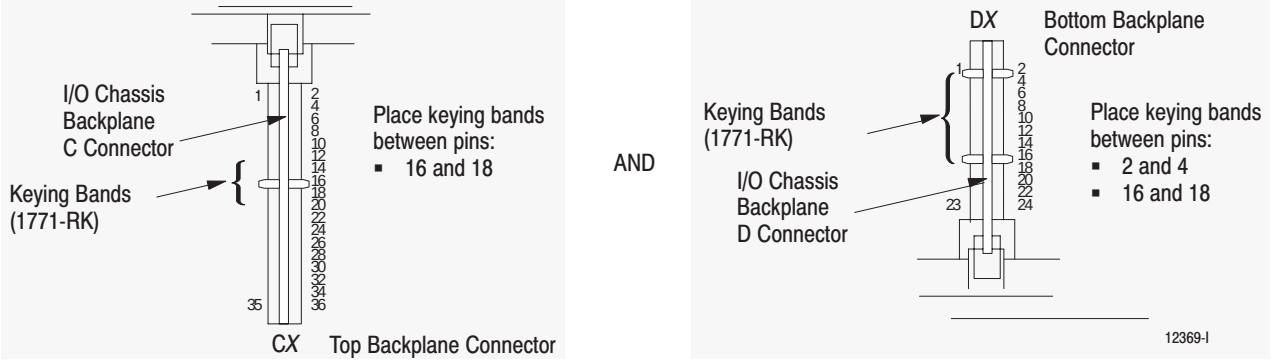


Figure 2.2
Keying Band Positions for the Serial Expander Module



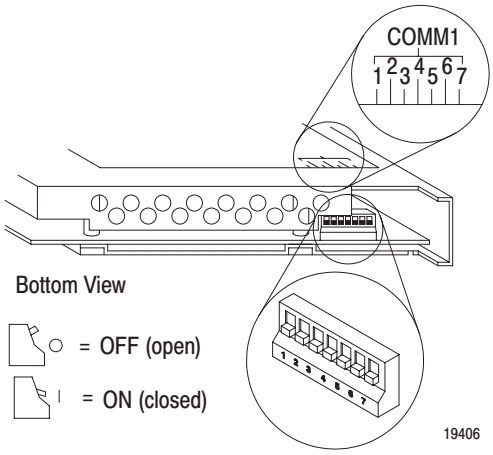
The serial expander module uses a total of three keying bands, two on the bottom connector and one on the top connector.

Set Switch Configurations for the Main Module

The COMM0 port has no switches to configure.

Set the COMM1 switches to configure the 25-pin asynchronous communication port.

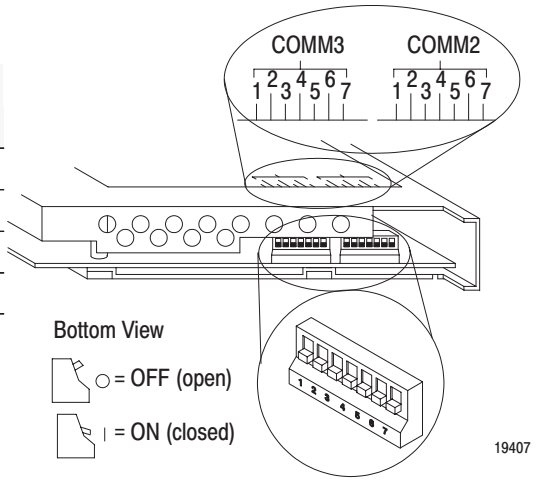
For this communication:	Set Switch						
	1	2	3	4	5	6	7
RS-232C	ON	ON	ON	OFF	OFF	ON	ON
RS-422	OFF	OFF	ON	OFF	OFF	OFF	OFF
RS-423	ON	ON	ON	OFF	OFF	ON	OFF
RS-485	ON	ON	ON	OFF	ON	ON	ON



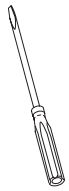
Set Switch Configurations for the Serial Expander Module

Set COMM2 and COMM3 switches to configure the 25-pin asynchronous communication ports.

For this communication:	Set Switch						
	1	2	3	4	5	6	7
RS-232C	ON	ON	ON	OFF	OFF	ON	ON
RS-422	OFF	OFF	ON	OFF	OFF	OFF	OFF
RS-423	ON	ON	ON	OFF	OFF	ON	OFF
RS-485	ON	ON	ON	OFF	ON	ON	ON



Install the Control Coprocessor



Install the control coprocessor in either direct-connect or standalone mode.

If you want to:	Then select:	On page:
Install a control-coprocessor main module directly connected to a PLC-5 programmable controller	Direct-Connect Installation	2-7
Install the optional serial expander module	Serial Expander Module Installation	2-9
Install a control coprocessor in the same chassis as, or remotely located from, a programmable controller but not directly connected	Standalone Installation	2-10

Direct-Connect Installation

For direct-connect installation, connect the control coprocessor to a PLC-5 programmable controller (with expansion port) using a connector header. Then, install the control coprocessor/PLC-5 programmable controller, as a unit, into an I/O chassis.

You need the following hardware:

- PLC-5 programmable controller with side connector
- PLC-5 connector header (1785-CNH/A)
- four screws
- four spacers
- ESD grounding wrist strap
- phillips screwdriver

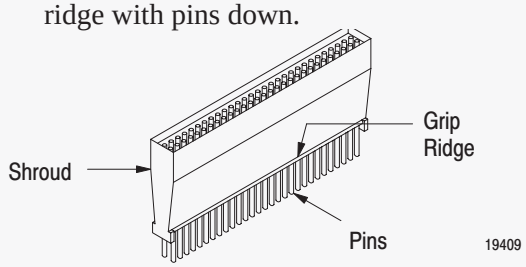
Connect Control Coprocessor to Programmable Controller

To connect the control coprocessor to the PLC-5 programmable controller:

ATTENTION: Avoid bending pins when installing the connector header into the PLC-5 programmable-controller side connector. Also, avoid bending pins when installing the control coprocessor onto the connector header.

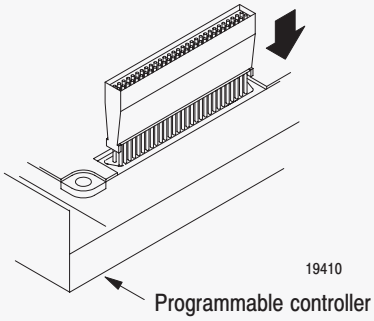
1. Place the PLC-5 programmable controller on a flat, anti-static surface with the side connector face up.

2. Hold the connector header at the grip ridge with pins down.



Important: It is important to attach the shroud and the pin side of the connector header in the designated device. However, the shroud side and the pin side are not keyed.

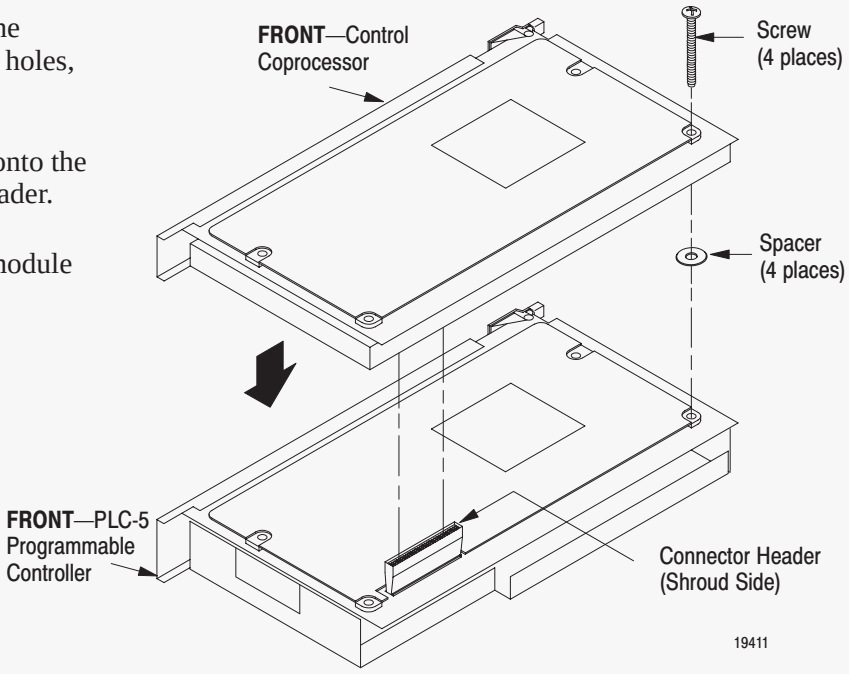
3. Install the connector header into the programmable controller.

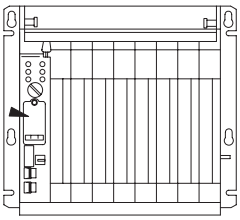


4. Place four nylon spacers over the programmable controller screw holes, adhesive side down.

5. Install the control coprocessor onto the shroud side of the connector header.

6. Install four screws, adjust the module alignment, and then tighten.

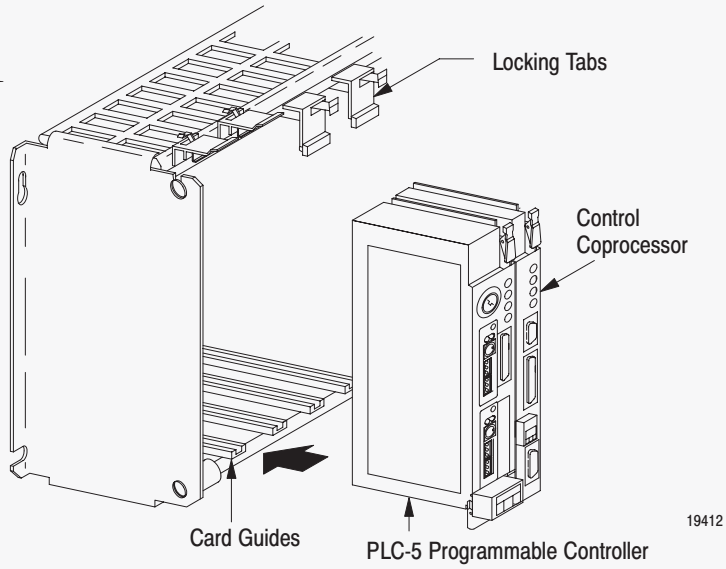




Install the Direct-Connect Control Coprocessor
To install the PLC-5 programmable controller and control coprocessor in the 1771 I/O chassis:

Important: If you are using the 1771 chassis with the locking bar rather than the locking tabs, refer to the Universal I/O Chassis Installation Data, publication 1771-2.210, for information on use.

1. Verify that power is **OFF** to the 1771 I/O chassis.
2. Install direct-connect modules in the I/O chassis using the left-most slots.
3. Slide until the modules fit into the backplane connectors.
4. Close the locking tabs.

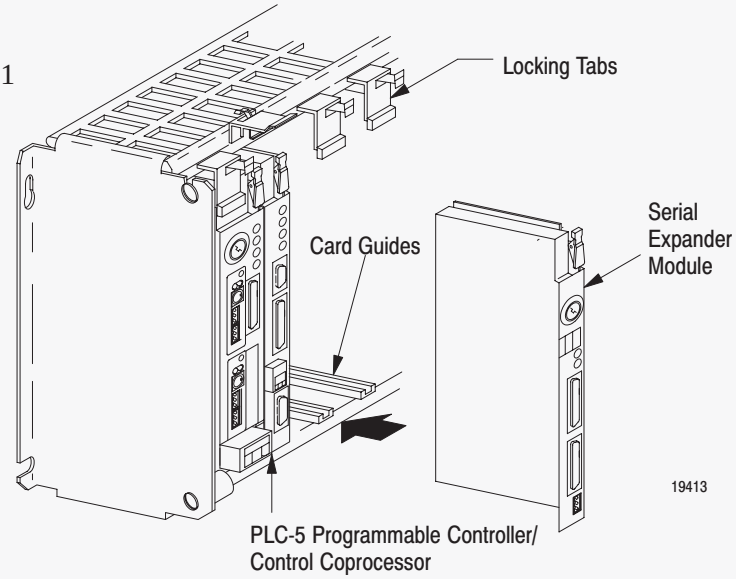


Serial Expander Module Installation

Install the serial expander module in the 1771 I/O chassis as follows.

1. Verify that power is **OFF** to the 1771 I/O chassis.
2. Slide the module into the I/O chassis in the slot immediately adjacent to the main module.
3. Slide until the module fits into the backplane connector.
4. Close the locking tab.

TIP The serial expander module must be in the same module group pair and under the same locking tab as the main module.



Standalone Installation

You can place the control coprocessor in any available slot in the I/O chassis with the following limitations:

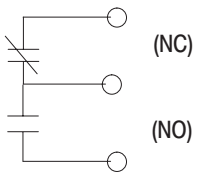
- We recommend that you configure the chassis for 1-slot addressing.
- The serial expander module, when used, must reside in the same module pair (under the same locking tab) as the main module.
- If you have two control coprocessors, place them in different module pairs. Two coprocessors cannot be placed under one locking tab.
- Place 1785-BCM/BEM backup communication module(s) in a different module pair (under a different locking tab) than the control coprocessor. You can place the 1785-BCM module in a slot adjacent to the control coprocessor but in a different module pair.
- Place 1771 I/O modules that require expander modules in a different module pair (under a different locking tab) than the control coprocessor. Examples are: 1771-IX, -QC, -QA, -OF, and -IF.

To install the control coprocessor in the 1771 I/O chassis:

1. Verify that power is **OFF** to the 1771 I/O chassis.
2. Using the card guides, insert the control coprocessor into the designated slot of the I/O chassis.
3. Slide the module until it fits into the chassis backplane connector.
4. Close the chassis locking tab for the module.
5. Install the serial expander module using the previous section, Serial Expander Module Installation, beginning on page 2-9.

Wire the Fault Relay

On the serial expander module, wire your load to the normally open (NO) or normally closed (NC) position, as appropriate for your application.



The fault relay is activated automatically when the main module faults or a main module is not adjacent to the serial expander module. A fault condition occurs when the control coprocessor's firmware cannot keep a hardware watchdog from timing out.

The fault relay can handle a load of 500 mA at 30 Vac/dc. You can use the fault relay for resistive loads without contact protection (to its rated load). For capacitive, inductive, filament, or other loads that produce surges, contact protection is recommended. Use relay manufacturer's data books to select contact protection devices or see the 1771 Discrete I/O

Relay Contact Output Modules Product Data, publication 1771-2.181, for more information.

Apply Power to the Control Coprocessor

The control coprocessor performs the following functions at power up:

- bootstrap routine
- OS-9 initialization
- A-B initialization (if direct-connect)
- invokes either a user start-up program or the OS-9 shell (command interpreter)
- hardware initialization (RAM disk, OS-9 clock, serial ports)
- fault-relay energizing

You will get the following normal indications on the main module after power up:

- CPU LED blinks green four times and then remains lit green
- BATT LED blinks red four times and then is not lit (this indicates a properly charged battery)

You will get the following normal indication on the optional serial expander module after power up: the four character positions on the ASCII display blink four times and then are not lit.

Remove the Control Coprocessor

When removing a main module in direct-connect or standalone mode, first verify that power is off to the 1771 I/O chassis; then, remove the module by reversing the installation procedure. If in direct-connect mode, remove the module from the PLC-5 programmable controller.

When removing a serial expander module, first verify that power is off to the 1771 I/O chassis; then, release the locking tab and remove the module from the I/O chassis.

What to Do Next

After you complete the installation and powerup of the control coprocessor, proceed to Chapter 3. Chapter 3 instructs you on how to connect a programming terminal to the control coprocessor and establish communication.

Chapter 3

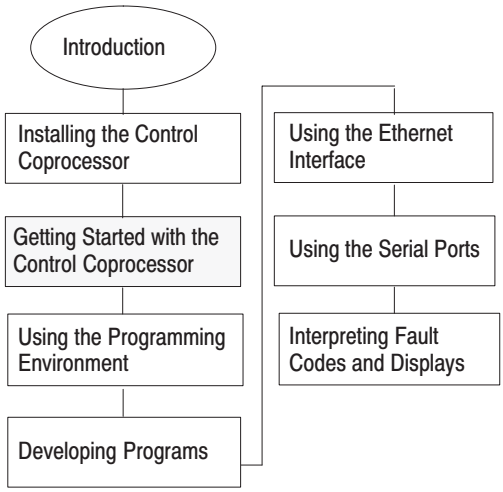
Getting Started with the the Control Coprocessor

Chapter Objectives

This chapter provides instructions on how to set up your control coprocessor for communication by:

- setting up your programming terminal
- setting up configuration parameters for the interface between the programming terminal and the control coprocessor
- testing the interface by completing the interface tasks

For information on:	See page:
Connecting the programming terminal	3-1
Selecting the programming interface	3-2
Installing 1771-PCB software	3-2
Accessing the PCBridge software	3-5
Configuring communication parameters	3-6
Accessing the OS-9 command-line interface	3-7
Configuring the control coprocessor	3-9
Viewing control coprocessor current status	3-19
Creating a user startup file	3-19
Sending a text file to the control coprocessor	3-20
Using other OS-9 commands	3-23



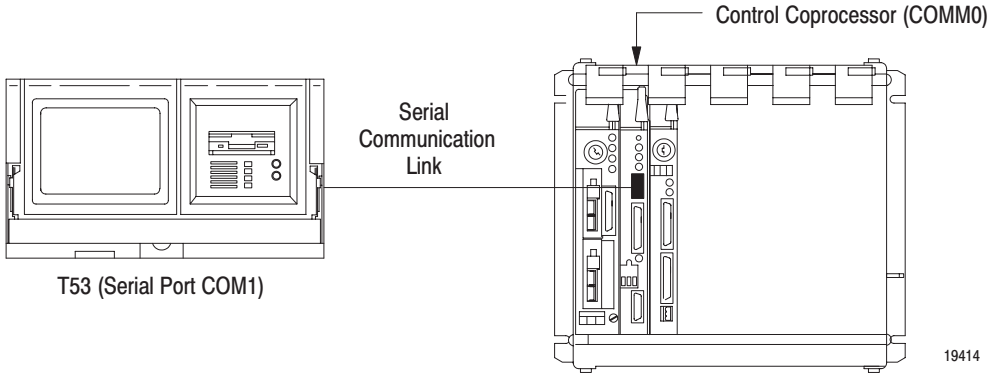
Connect the Programming Terminal

You can program the control coprocessor via a personal computer or an ASCII terminal.

Personal Computer (DOS-Based) Terminals	ASCII Terminals
IBM PC/AT™	VT220™ (DEC)
Allen-Bradley T47, T50, T53, or T60	Other ASCII terminals

Connect the programming terminal to the COMM0 port—default terminal port—of the main module. See Appendix C for cable and connector information.

Figure 3.1
Personal Computer to Control Coprocessor Connection



Select the Programming Interface

The programming terminal that you select determines how you program the control coprocessor.

If you use:	You program using:	See page:
A personal computer	the PCBridge software and your text editor	3-2
An ASCII terminal	OS-9 command-line interface	3-7

Install the Software on Your Personal Computer

Before you install and use the PCBridge software (1771-PCB), you must have a working knowledge of DOS and its utilities, such as: DIR, COPY, and TYPE. You must also be able to use a DOS text editor.

To install the PCBridge software and the library of functions you will use for your programming, your personal computer must have at least:

- 640 Kbytes RAM
- 2 Mbytes online disk storage
- DOS 4.0 or later

Included with the PCBridge software is:

- a C cross-compiler, cross-assembler, and cross-linker
- a C source-code debugger
- Kermit—for sending/receiving data files and application programs
- Internet-support software
- A-B interface libraries
- a text editor

See Appendix D for more information. See Chapter 5 and Appendix B for more information on the A-B interface libraries.

To install the software:

1. Insert the first disk.

2. At the DOS prompt, type **`install`** ***`dest_drive`***: and press **`[Return]`**.

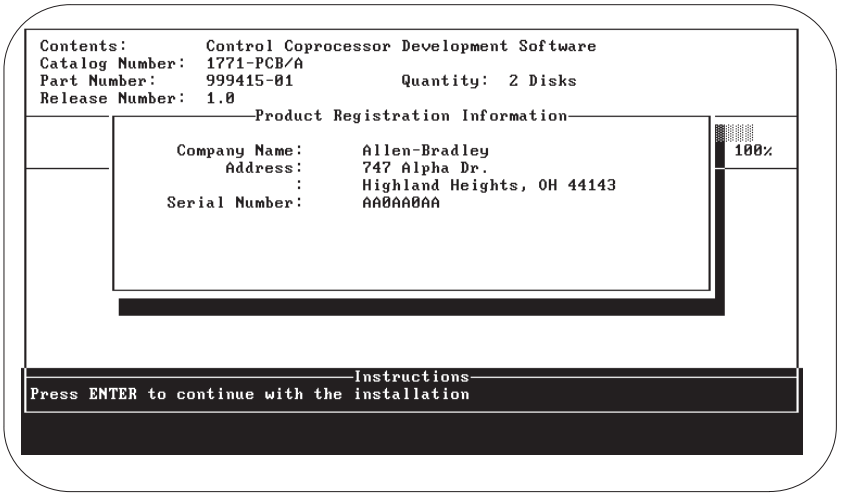
For example: if your source drive is b: and you want to install the software on your hard drive c:, then type;

```
b:install c:
```

and press **`[Return]`**. The software displays a screen regarding the licensing agreement and copyright protection.

3. Press **`[Return]`**.

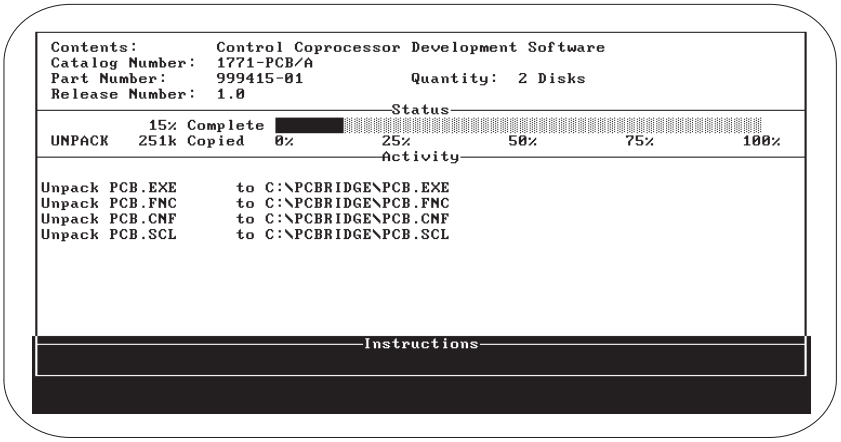
Figure 3.2
Registration Screen



4. Fill in your company name (up to 31 characters) and address as well as the serial number of your software.

5. Press **`[Return]`** to save. You get the screen to install your software.

Figure 3.3
Download the PCBridge Software



6. Press **[Return]** to begin installing the software. The percentage-complete graph increments as the software is loaded.

7. Install the remaining disk(s) when the system prompts you.

DIRECTORY STRUCTURE

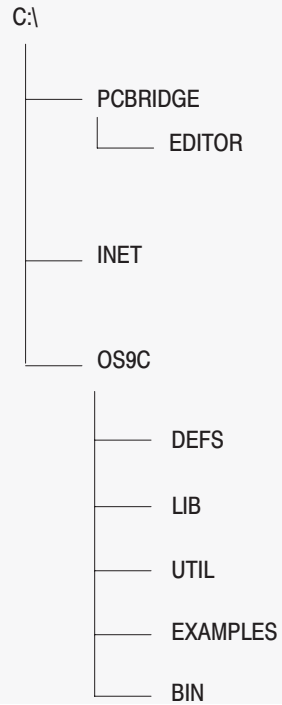
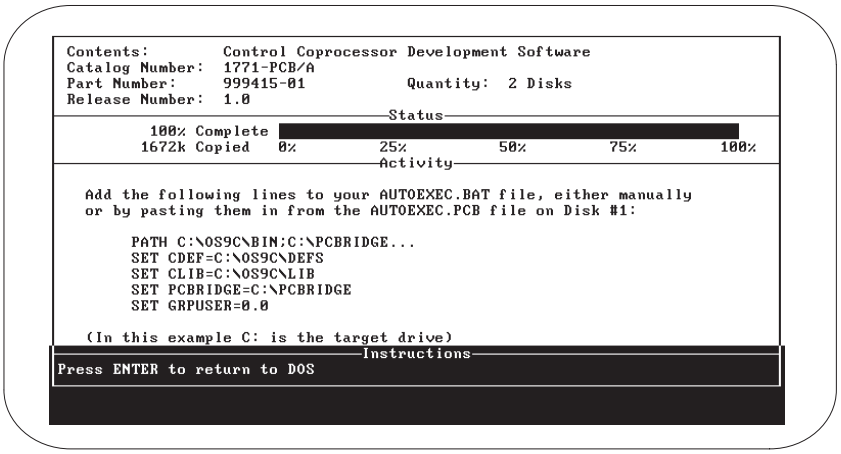


Figure 3.4
Files to Modify



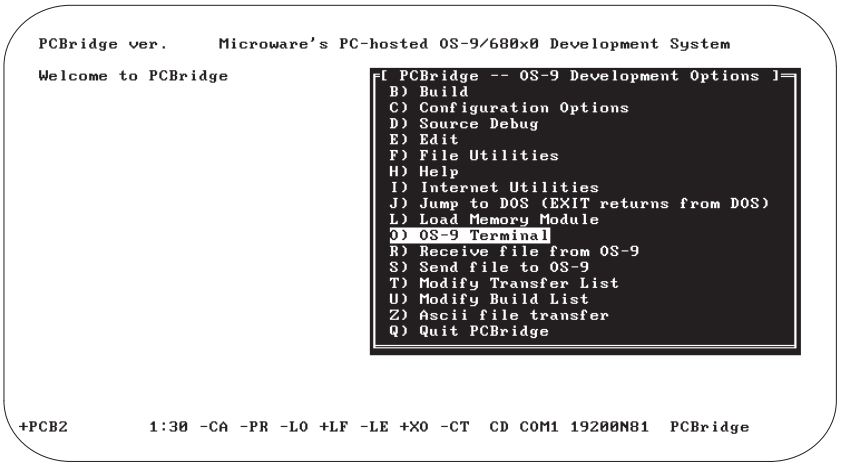
Important: After you have installed the disks, the system informs you of any files that you must modify—e.g., AUTOEXEC.BAT, PCB.CNF—to enable the compiler to start properly.

Do not use the NOEMS flag as part of the EMM386.EXE command in the CONFIG.SYS file. Instead, use the RAM flag, which allows the PCBridge software and other applications to use both extended and expanded memory.

Access the PCBridge Software

Access the PCBridge software from the DOS command line by typing **pcb** and pressing **[Return]**. See Figure 3.5.

Figure 3.5
PCBridge Main Menu



Note that the +PCB2 line at the bottom of the screen is a status line. Among other information that it provides, it informs you of the status of the link between the personal computer and the control coprocessor.

Tip

To select options from the main menu, use the arrow keys to cursor to a choice on the menu and press **[Return]**; or you can simply type the letter of your choice.

Use any of the following methods to highlight a menu item:

- use the **[↑]** or **[↓]** cursor keys to move the highlighting up or down
- use **[Space Bar]** to move the highlighting to the next item on the menu
- enter the first character of a menu item to select it

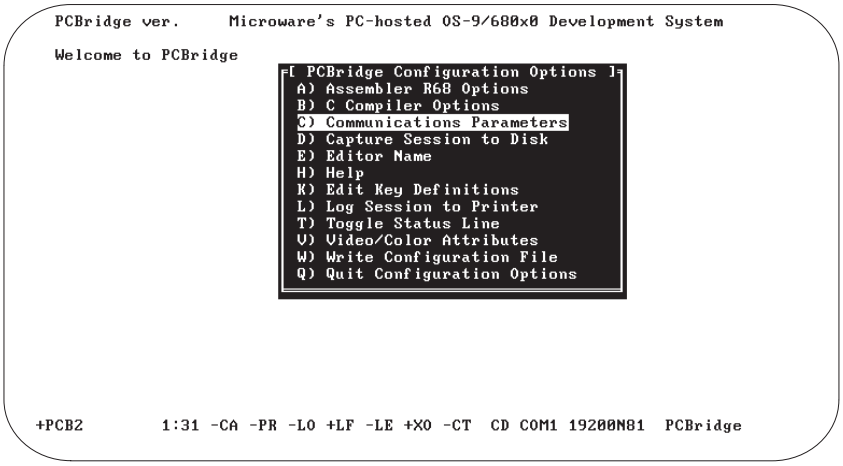
Many of the menu items, when selected, prompt you for further information. Most screens allow you to exit and stop execution of the option if you press **[Esc]** before you press **[Return]**. This aborts the operation and returns you to the previous screen.

Configure Communication Parameters

To configure parameters for the communication interface between the personal computer and the control coprocessor:

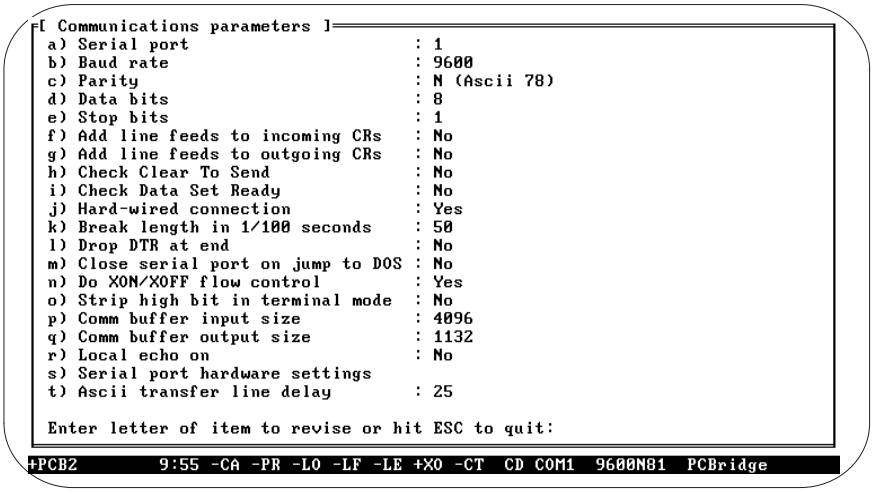
1. Select `C) Configuration Options` on the PCBridge main menu. You get the PCBridge Configuration Options screen. See Figure 3.6.

Figure 3.6
PCBridge Configuration Options Screen



2. On the PCBridge Configuration Options menu, select `C) Communication Parameters`. See Figure 3.7.

Figure 3.7
PCBridge Communication Parameters Screen



3. On the PCBridge Communications Parameters screen, select the parameters for communication with the control coprocessor. These parameters are the default setup of the control coprocessor:

   - 9600 baud
   - no parity
   - 8 data bits
   - 1 stop bit

   See Appendix A, Control-Coprocessor Specifications, for other available rates of communication.

   Important: If you want to change the communication rate for the personal computer via the PCBridge software, you must first change the communication rate for the control coprocessor. See the section on creating a user startup file—page 3-19—for more information.

   Select each parameter that you want to change. You get a menu with options available for that parameter. Select the option for your application.

4. After entering all of your new parameters, press **[Esc]** to quit the screen and return to the configuration menu.

5. Select `W) Write Configuration File` to save your communication configuration.

6. Press **[Esc]** to quit the screen and return to the PCBridge main menu.

Access and Use the OS-9 Command-Line Interface

To use the OS-9 command-line interface:

1. Press **[Return]** on your ASCII terminal, or select `O) OS-9 Terminal` and press **[Return]** on the PCBridge main menu, to access the OS-9 command-line interface.

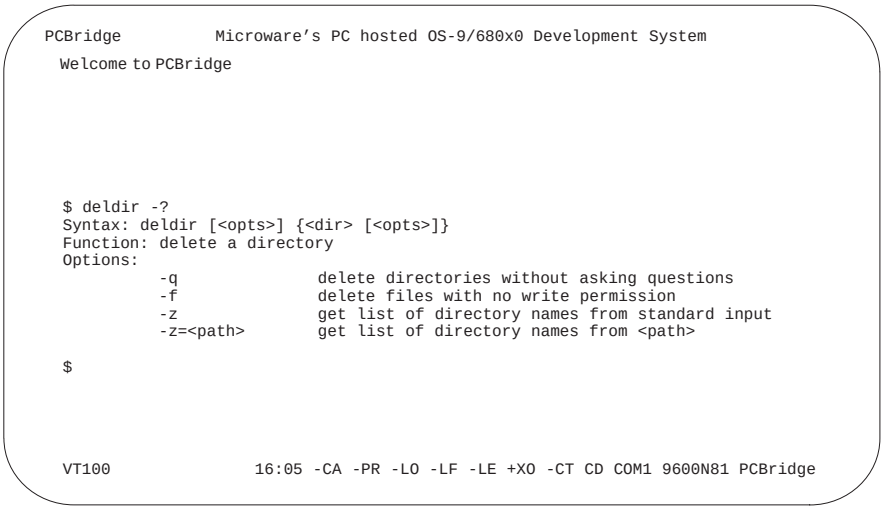
2. At the `$` prompt, type any of the available OS-9 standard utilities and built-in shell commands.

See the OS-9 Operating System User Manual, publication 1771-6.5.102, for more information.

Get Help for OS-9 Utilities

At the `$` prompt, type the name of the utility for which you want more information, followed by space `-?` and press **[Return]**. See the example in Figure 3.8. You get information on the syntax, function, and options for that utility.

Figure 3.8
OS-9 Command-Line Interface Utility Help

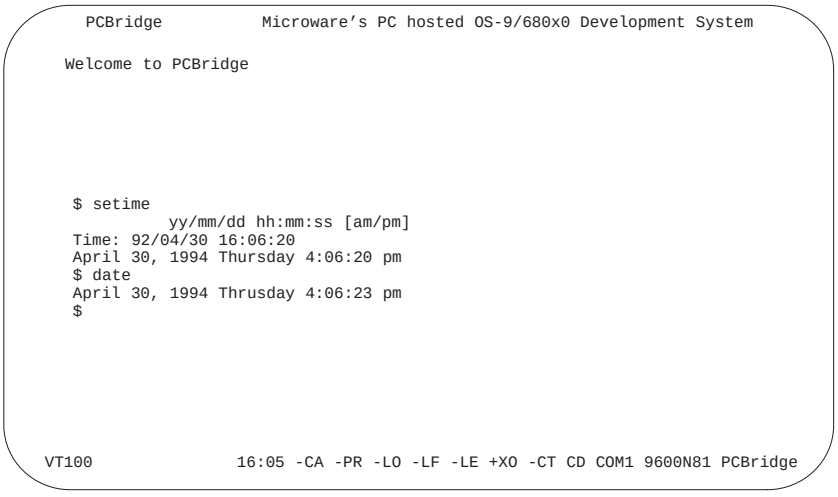


At the $ prompt, type **mdir** and press **[Return]** for a list of all available utilities.

Set Time for OS-9

At the **$** prompt, type **setime** and press **[Return]** to set the time and date for the control-coprocessor operating system. See Figure 3.9.

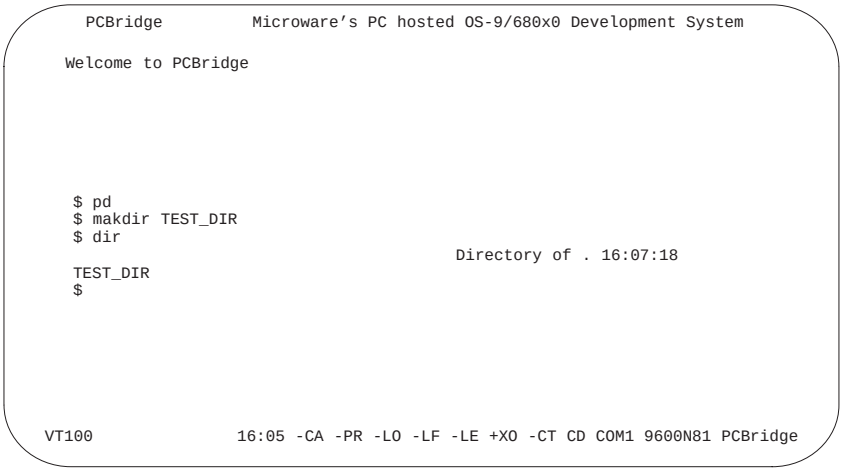
Figure 3.9
OS-9 Command-Line Interface Setime Utility



Create a Test Directory

At the $ prompt, type **`makdir`** followed by a space and the name of the test directory that you want to create; then, press **`[Return]`**. Change your working directory to the one that you just created. See Figure 3.10.

Figure 3.10
OS-9 Command-Line Interface Make Directory and Change Directory



Return to the PCBridge Main Menu from the OS-9 Command Line

Press **`[F1]`** to return to the PCBridge main menu from the OS-9 command-line interface. From the PCBridge main menu, you can select other PCBridge options.

Configure the Control Coprocessor

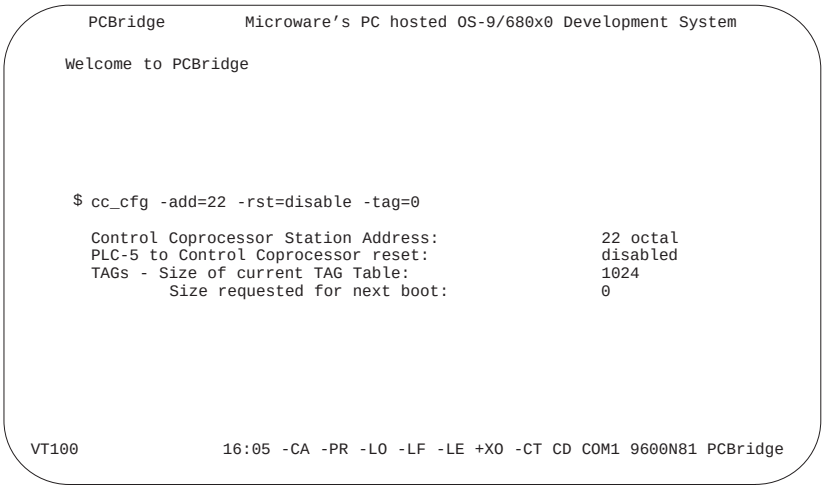
You must do the following to configure the control coprocessor:

To configure:	See page:
Default startup parameters (CC_CFG)	3-10
System memory (MEM_CFG)	3-11

Configure the Default Startup Parameters

Configure the default startup parameters of the control coprocessor using the CC_CFG utility. See Figure 3.11.

Figure 3.11
Default Parameters for the Control Coprocessor



With this utility, you:

- set the control coprocessor station address
- enable/disable the capability of the PLC programmable controller to reset the control coprocessor when the PLC programmable controller encounters a hardware fault
- configure the size of the TAG table

Syntax for the CC_CFG utility is:

```
cc_cfg {<opts>}
functions: Configure Control Coprocessor Parameters
options:   -add=<octl num> Station Address (0-77 octal)
           -rst=<str>      Enable/Disable PLCs' reset to CC
           -tag=<num>      Request size of TAG table
```

The station address and reset parameters take effect immediately.

The selected size of the TAG table is effective after the next system boot. If there is insufficient memory available for the configuration, 1024 is the default size used for 1771-DMC1 and 1771-DMC4 control coprocessors. Zero is the default TAG-table size for the 1771-DMC control coprocessor.

Configure System Memory

Configure the control coprocessor system memory using the MEM_CFG utility. You can configure the size of the following non-volatile memory sections:

- RAM disk—page 3-12
- user memory—page 3-13
- module memory—page 3-15

See Figures 3.12 through 3.20 for an example using the MEM_CFG utility.

Figure 3.12
Memory Configuration

```
PCBridge    Microware's PC hosted OS-9/680x0 Development System

$ MEM_CFG
                Control Coprocessor Memory Configuration Utility

                                    Original     Current
                                    Settings     Settings
------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         64Kb
Non-Volatile User Memory    =        0Kb          0Kb
Non-Volatile Module Memory  =        0Kb          0Kb
OS-9 Free Pool              =     4800Kb       4800Kb
                                  ----------   ----------
Configurable System Memory  =     4864Kb       4864Kb

Main Menu Selection
-------------------
1 = Configure Non-Volatile RAM Disk Size
2 = Configure Non-Volatile User Memory Size
3 = Configure Non-Volatile Module Memory Size
4 = Configure System (reboot)

Select Option:
```

You can allocate all of the system RAM to the non-volatile memory sections except for 128 Kbytes that are allocated for the control coprocessor, OS-9 operating system, and the free-memory pool of the operating system. Any RAM that you do not configure as non-volatile is allocated to the operating system's free-memory pool.

The non-volatile module memory (NVMM) utility controls the non-volatile module memory. Use the NVMM utility to manage your program modules in memory. See page 3-16.

After you make all your changes to the memory configuration, you must select option 4 from the main menu. See page 3-18. This reboots the system and activates your changes.

ATTENTION: If configuring the memory results in an out-of-memory error, you can recover the default memory setup by removing the battery from the coprocessor for several minutes.

RAM Disk

The RAM disk is an emulated drive that resides in Random Access Memory (RAM). You can store and access any files on a RAM disk. The default size of the non-volatile RAM disk is 64 Kbytes.

ATTENTION: Changing the size of the non-volatile RAM disk will reformat it. Back up the disk data before changing the RAM-disk size.

To configure the non-volatile RAM disk:

1. At the `Select Option` prompt, enter **`1`** for the non-volatile RAM-disk configuration option. See Figure 3.13.

Figure 3.13
Configure the RAM Disk

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System


Select Option: 1
                                  Original     Current
                                  Settings     Settings
-------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         64Kb
Non-Volatile User Memory    =        0Kb          0Kb
Non-Volatile Module Memory  =        0Kb          0Kb
OS-9 Free Pool              =     4800Kb       4800Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb       4864Kb

Non-Volatile RAM Disk Size
--------------------------
Enter desired number of 64K (65536) byte blocks for disk (1- 76): 8

                                  Original     Current
                                  Settings     Settings
------------------------------------------------------------- -
Non-Volatile RAM Disk       =       64Kb        512Kb
Non-Volatile User Memory    =        0Kb          0Kb
Non-Volatile Module Memory  =        0Kb          0Kb
OS-9 Free Pool              =     4800Kb       4352Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb       4864Kb
```

2. At the prompt, enter the number of blocks—between 1 and the maximum amount as shown by the utility—that you want to allocate to the RAM-disk size.

Non-Volatile User Memory

This is a non-volatile area of memory that is not known to the operating system; therefore, any data stored here remains intact through resets and power cycles. This non-volatile memory is controlled by user programs.

This area of memory is basically a storage area for data. Although you can use it for any purpose, one common application is to use this area as a common memory area for multiple programs—this makes effective use of the fact that this memory is non-volatile.

1. At the `Select Option` prompt, enter **`2`** for the non-volatile user-memory configuration option. See Figure 3.14.

Figure 3.14
Configure Non-Volatile User Memory

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System
Select Option: 2
                                  Original     Current
                                  Settings     Settings
------------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         512Kb
Non-Volatile User Memory    =        0Kb           0Kb
Non-Volatile Module Memory  =        0Kb           0Kb
OS-9 Free Pool              =     4800Kb        4352Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb        4864Kb

Non-Volatile User Memory Size
-----------------------------
Enter desired number of 1K (1024) byte blocks (0 - 4352): 5

                                  Original     Current
                                  Settings     Settings
------------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         512Kb
Non-Volatile User Memory    =        0Kb           5Kb
Non-Volatile Module Memory  =        0Kb           0Kb
OS-9 Free Pool              =     4800Kb        4347Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb        4864Kb
```

2. At the prompt, enter the number of blocks that you want to allocate to non-volatile user memory—between 0 and the maximum amount as shown by the utility.

ATTENTION: Do not change the pointer values. They are intended to be read-only. Subsequent memory configurations can change the pointer value to the start of the non-volatile user memory. User programs must contain comparisons to check that the pointer value has not changed from the originally stored value.

See the following example program—MY_MEM.C. Address 0x10000200 contains a pointer to the start of the non-volatile user memory. Address 0x10000204 contains the size of the block in bytes (an unsigned integer or 4 bytes). The control coprocessor sets the data at addresses 0x10000200 and 0x10000204, dependent on memory configuration.

Non-Volatile Memory Example User Program MY_MEM.C

```
#include <time.h>                                /* include time header file */
#define  PM_PTR 0x10000200       /* this is where my nv memory ptr is stored */
extern time_t time();                              /* function declarations */
struct tm *localtime();
char * asctime();
typedef struct                               /* define my structure in nv ram */
  {
  unsigned *ptr_check;                     /* storage for checking nv pointer */
  char  time_stamp[26];                        /* xxx mmm dd hh:mm:ss yyy\n\0 */
  unsigned count;                                          /* boot count */
  }MY_MEM;
main (argc, argv)
 int argc;
 char *argv[];
{
  MY_MEM *mm_ptr;                                     /* ptr to my memory */
  char *tim_ptr;                                /* string ptr for time string */
  time_t cal_time;                                 /* calendar time storage */
  struct tm *loc_time;                                /* local time storage */
  mm_ptr = *(MY_MEM **)PM_PTR;                  /* get pointer to my nv data */
  if (mm_ptr == 0)                                /* make sure its allocated */
     {
      printf (”Protected Memory not allocated\n”);
      exit (0);
     }
  if (argc > 1)mm_ptr->ptr_check=(unsigned *)mm_ptr;     /* store ptr on init */
  else
 {
       if (mm_ptr != (MY_MEM *)mm_ptr->ptr_check)     /* check if ptr changed */
          {
           printf (”Protected Memory Pointer changed\n”);
           exit (0);
          }
      }
  cal_time = time(0);                                          /* get time */
  loc_time = localtime (&cal_time);                  /* convert to local time */
  tim_ptr = asctime (loc_time);                       /* convert to string */
  if (argc > 1)            /* if command line parameter then initialize data */
      {
      mm_ptr->count = 0;                                /* start count at 0 */
      strncpy (mm_ptr->time_stamp,tim_ptr,26);          /* store initial time */
      printf (”\nCurrent time is ->%s\n”,mm_ptr->time_stamp);
      }
  else
      {
      printf (”\nTime of last boot ->%s\n”,mm_ptr->time_stamp);  /* print old */
      strncpy (mm_ptr->time_stamp,tim_ptr,26);         /* copy new time to nv */
      printf (”Time of this boot ->%s\n”,mm_ptr->time_stamp);    /* print new */
      mm_ptr->count +=1;                               /* increment boot count */
      printf (”Boot count = %d\n”,mm_ptr->count);         /* print boot count */
      }
}
```

ATTENTION: This program illustrates the use of non-volatile user memory in its simplest form. The control-coprocessor MEM_CFG function only supplies a pointer and size to the non-volatile user memory. It is the responsibility of the user to manage the memory appropriately.

The program stores the value of the pointer on initialization. It then performs subsequent checks to verify that the pointer value has not changed.

Non-Volatile Module Memory
Use this non-volatile area of memory to store program modules. Although you can store your programs on the non-volatile RAM disk, the modules must also be loaded to OS-9 memory to run. When you store them in the NVMM area, the modules are in a ready-to-run state and do not use memory on the RAM disk unnecessarily.

This non-volatile memory is non-destructively searched at boot by the operating system for program modules. Reset or power-down conditions will not destroy modules in this memory area.

To configure the memory area in 1 Kbyte (1 block) increments:

1. At the `Select Option` prompt, enter **3** for the non-volatile memory-module configuration option. See Figure 3.15.

Figure 3.15
Configure Non-Volatile Memory Modules

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System
Select Option: 3
                                  Original     Current
                                  Settings     Settings
-------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         512Kb
Non-Volatile User Memory    =        0Kb           5Kb
Non-Volatile Module Memory  =        0Kb           0Kb
OS-9 Free Pool              =     4800Kb        4347Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb        4864Kb

Non-Volatile Module Memory Size
-------------------------------
Enter desired number of 1K (1024) byte blocks (0 - 4347): 70

                                  Original     Current
                                  Settings     Settings
-------------------------------------------------------------
Non-Volatile RAM Disk       =       64Kb         512Kb
Non-Volatile User Memory    =        0Kb           5Kb
Non-Volatile Module Memory  =        0Kb          70Kb
OS-9 Free Pool              =     4800Kb        4277Kb
                                 ----------   ----------
Configurable System Memory  =     4864Kb        4864Kb
```

2. At the prompt, enter the number of blocks that you want to allocate to non-volatile memory modules—between 0 and the maximum amount as shown by the utility.

NVMM Utility

With the NVMM utility, you can:

- move modules from OS-9 to non-volatile module memory
- list all modules in the non-volatile module memory
- enable deletion of modules in the non-volatile module memory
- delete modules from the non-volatile module memory

Syntax for the NVMM utility is:

```
NVMM -M [module]      Moves module into
                      non-volatile module memory
NVMM -L               Lists all modules in
                      non-volatile module memory
NVMM -D               Enables deletion of modules in
                      non-volatile module memory
```

Important: When you move modules to the non-volatile module memory, they are not included in the OS-9 Module Directory until the next system boot.

When `delete enable` is set by the NVMM utility, the next system boot automatically invokes NVMM in a menu mode. From the menu, you can list and delete modules in the non-volatile module memory.

To protect modules that are used by other processes, you can delete the modules from non-volatile module memory only during the next system boot.

See Figure 3.16 through Figure 3.18 for an example session of NVMM.

Figure 3.16
NVMM Session (1 of 3)

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System



$ NVMM -M MY_MEM
Non-Volatile Module Memory Move Utility
Module [MY_MEM] moved to Non-Volatile Module Memory
$
$ NVMM -L
Non-Volatile Module Memory List Utility
ADDRESS       SIZE HEX     SIZE DECIMAL     MODULE NAME
------------------------------------------------------------------------
101ee800          556H             1366     hello
101eed56          74cH             1868     my_mem
Total size of Non-Volatile Module Memory is    71680 ( 11800h) bytes
Largest contiguous Non-Volatile memory is      68446 ( 10b5eh) bytes
$
$ NVMM -D
Non-Volatile Module Memory Delete is enabled
```

Figure 3.17
NVMM Session (2 of 3)

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System


$
Non-Volatile Module Memory Menu Selection
-----------------------------------------
1 = Delete module in Non-Volatile Module Memory
2 = Delete all modules in Non-Volatile Module Memory
3 = List all modules in Non-Volatile Module Memory
4 = Exit (continue boot)
Select Option: 3
Non-Volatile Module Memory List Utility
ADDRESS       SIZE HEX     SIZE DECIMAL     MODULE NAME
-------------------------------------------------------------------------------
101ee800          556H             1366     hello
101eed56          74cH             1868     my_mem
Total size of Non-Volatile Module Memory is    71680 ( 11800h) bytes
Largest contiguous Non-Volatile memory is      68446 ( 10b5eh) bytes
```

Figure 3.18
NVMM Session (3 of 3)

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System

$
NVMM Menu Selection
-------------------
1 = Delete NVMM Module
2 = Delete all NVMM Modules
3 = List all NVMM Modules
4 = Exit (continue boot)
Select Option: 1
Non-Volatile Module Memory Delete Utility
Enter module name: hello
Module [hello] deleted
NVMM Menu Selection
-------------------
1 = Delete NVMM Module
2 = Delete all NVMM Modules
3 = List all NVMM Modules
4 = Exit (continue boot)
Select Option: 4
                    Allen-Bradley Control Coprocessor
                Copyright 1994, Allen-Bradley Company, Inc.
                          All Rights Reserved
                      Series/Revision A/E (1.30)
$
```

MY_MEM displays the time of the last boot, time of this boot, and the boot count. The boot count and time of last boot are stored in non-volatile user memory. The startup file, STARTUP, includes the line `MY_MEM` to invoke the program at system boot.

See Figure 3.19 for an example MY_MEM boot screen. See page 3-14 for the source file, MY_MEM.C.

Figure 3.19
MY_MEM Boot Screen

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System

$MY_MEM
$
Time of last boot ->Wed Jul 22 12:23:03 1994
Time of this boot ->Wed Jul 22 18:12:08 1994
Boot count = 3
                        Allen-Bradley Control Coprocessor
                   Copyright 1994, Allen-Bradley Company, Inc.
                              All Rights Reserved
                          Series/Revision A/E (1.30)
```

Reboot to Configure System

After you make all your memory configuration changes, select option 4 from the main menu to reboot and invoke the changes that you made. See Figure 3.20.

Important: You must select option 4 on the main menu to activate any changes that you make. **Option 4 reboots the system.** Use [Ctrl-C] to abort this utility at any time and cancel any requested changes.

Figure 3.20
Configure System (Reboot)

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System
  Main Menu Selection
  -------------------
  1 = Configure Non-Volatile RAM Disk Size
  2 = Configure Non-Volatile User Memory Size
  3 = Configure Non-Volatile Module Memory Size
  4 = Configure System (reboot)
  Select Option: 4
                        Allen-Bradley Control Coprocessor
                   Copyright 1994, Allen-Bradley Company, Inc.
                              All Rights Reserved
                          Series/Revision A/E (1.30)
                   Warning: Memory configuration has changed
                   ========================================
  User Startup File "/dd/startup"  bypassed
  Non-Volatile RAM Disk configuration has changed
  Non-Volatile User Memory configuration has changed
  Non-Volatile Module Memory configuration has changed
  Pointer to Non-Volatile User Memory has changed
    Current pointer to Non-Volatile User Memory:  101ed400
    Current size of Non-Volatile User Memory:           5k
    Current size of Non-Volatile Module Memory:        70k
    Current size of Non-Volatile RAM Disk:            512k
  Use the NVMM -L utility to verify contents of Non-Volatile Module Memory
```

View Control-Coprocessor Current Status

Use the CC_STATUS utility to view the current status of the control coprocessor. See Figure 3.21 for an example screen.

Figure 3.21
CC_STATUS Screen

```
PCBridge        Microware's PC hosted OS-9/680x0 Development System

$ CC_STATUS

Allen-Bradley Control Coprocessor Status

Series/Revision: ............. A/E (1.30)
PLC-5: ....................... on-line
Expander: .................... not present
Battery: ..................... OK
PLC-5 to Coprocessor Reset: ... disabled
Station Address: ............. not set
Ethernet Address: ............ 00 00 BC 1B 00 0A
Total Memory: ................ 5120k
TAG Table: ................... 1024
Non-Volatile Memory
   RAM Disk: ................. 640k
   Module Memory: ............ 50k
   User Memory: .............. 3k
```

Create a User Startup File

When you power up or reset the control coprocessor, it executes the startup file /DD/STARTUP. This file is a text file that contains one or more command lines. The shell executes each command line in the exact order given in the file. It is similar to a DOS AUTOEXEC.BAT file.

You can bypass the execution of the startup file by holding the control coprocessor reset button—or the keyswitch on the serial expander module—until the CPU LED on the main module begins to blink, approximately 5 seconds.

In order for the startup file to execute on powerup or reset, you must have previously executed the `setime` command to set the real-time clock.

Example Startup File

The following is an example of a startup file;

```
tmode -w=0 -w=1 baud=19200   *change baud of /term port
xmode /t1 baud=19200         *change baud of /t1 port
shell <>>>/t1&               *activates a shell on /t1
procs                        *see what processes are currently running
```

You cannot set environment variables in a startup file because OS-9 invokes a separate shell to run the script file. However, you can set environment variables when you set up a password file. See page 3-20 for more information.

See OS-9 Operating System User Manual, publication 1771-6.5.102, for more information on shell procedure files.

Set Up a Password File

After the control coprocessor executes the startup file, it executes the login file. This file must have the appropriate entries for the login to execute. If the control coprocessor does not find the DD/SYS/PASSWORD file, it executes the OS-9 shell.

Important: When using Ethernet, you must have a password file in the /DD/SYS directory. When you are not using Ethernet, the password file is optional.

The password file contains one or more variable-length text entries—an entry for each user name. The fields are separated by commas and defined as follows:

- user name
- password
- group.user ID number
- initial process priority
- initial execution directory pathlist
- initial data directory pathlist
- initial program

The following is an example of a password file:

```
super,user,0.0,255,.,.,shell -p="Super:"
fudja,ajduf,3.7,128,/dd,/dd,shell
```

Set your environment variables in a .LOGIN file. The .LOGIN file is executed when the /DD/SYS/PASSWORD file is present on the RAM drive and the user is forced to log in.

See OS-9 Operating System User Manual, publication 1771-6.5.102, for more information on password files and execution of the login procedure.

Send a Text File to the Control Coprocessor

This section explains how you create a text file and then send it to the control-coprocessor RAM disk.

Create a Test Text File

To create a text file:

1. Select `E) Edit` on the PCBridge main menu to access your editor.

   The default text editor is DTE, a public domain text editor provided for your convenience.

2. Create a text file. The test file for this example is named TEST.TXT.

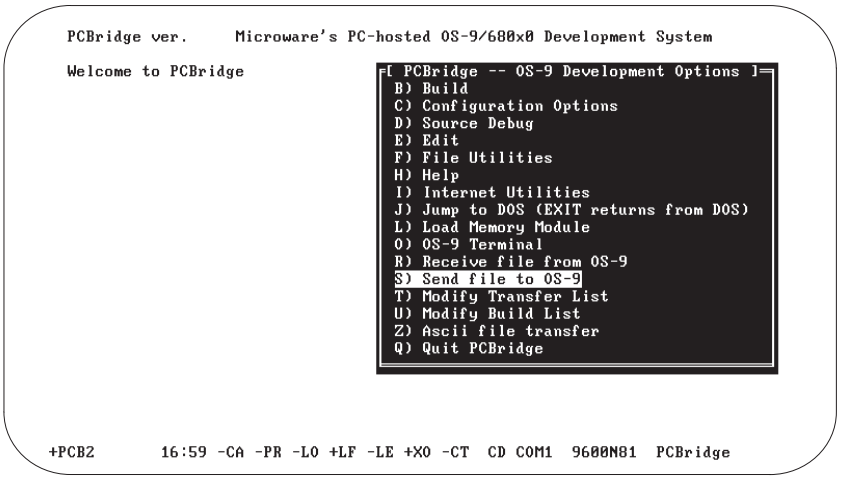
3. Return to the PCBridge main menu after you complete writing your text file.

Send the Text File to OS-9

To send the TEST.TXT file to the OS-9 RAM disk:

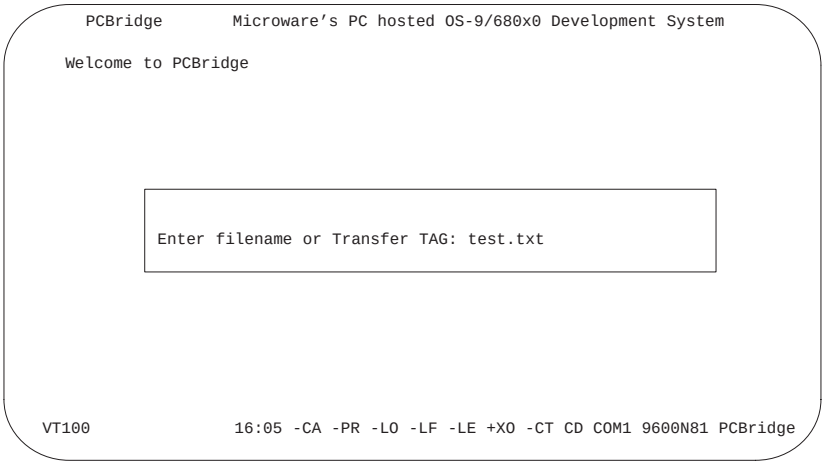
1. Select `S) Send file to OS-9` on the PCBridge main menu. See Figure 3.22.

Figure 3.22
Select Send File on Main Menu



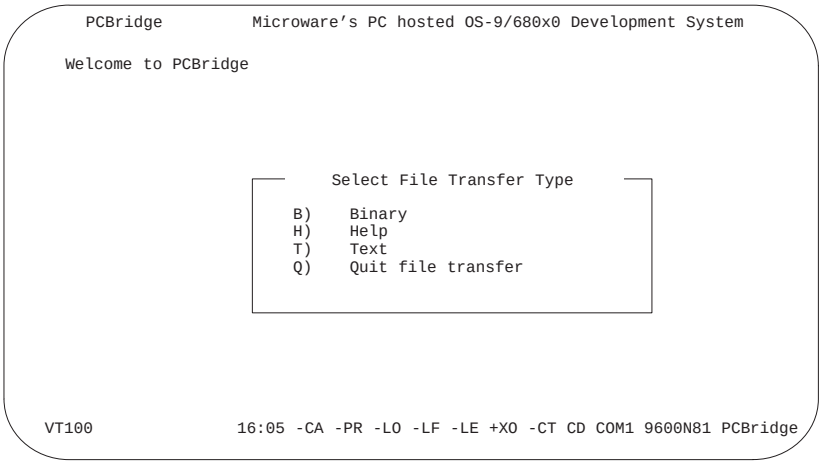
2. Enter the name of your text file in the prompt window. See Figure 3.23.

Figure 3.23
Enter Name of Test Text File to Send to OS-9



3. Select the file transfer type `T) Text`. See Figure 3.24.

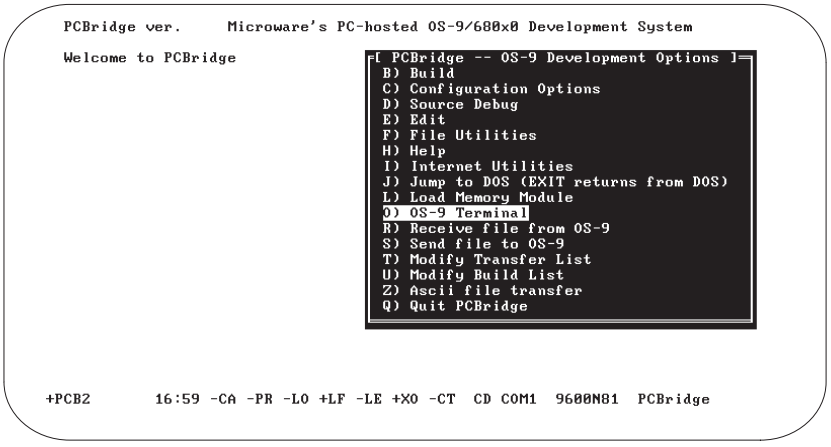
Figure 3.24
Select File Transfer Type



The PCBridge software automatically invokes Kermit and downloads the text file.

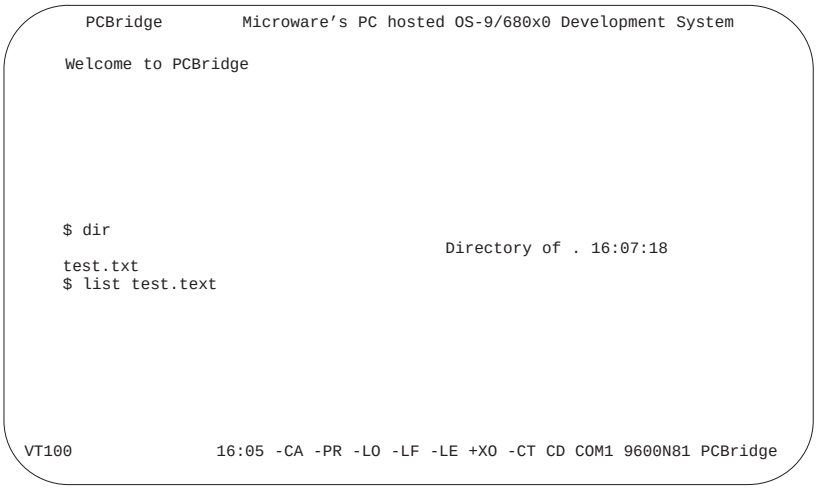
4. Select `O) OS-9 Terminal` on the PCBridge main menu. You get the control coprocessor OS-9 command-line interface. See Figure 3.25.

Figure 3.25
Select OS-9 Terminal on Main Menu



5. At the $ prompt, type **`dir`** and press **`[Return]`**. Observe that the text file was successfully transferred to the RAM disk . See Figure 3.26.

Figure 3.26
Check Directory for Test File



6. At the $ prompt, type **`list test.txt`** and press **`[Return]`**. The contents of the file are typed to the screen. Note that `list` is the OS-9 equivalent of the MS-DOS type command.)

Find Other OS-9 Commands

See the OS-9 Operating System User Manual, publication 1771-6.5.102, for information on other OS-9 commands and utilities.

What to Do Next

Proceed to Chapter 4. In Chapter 4, you create sample BASIC and C programs and download them to the control coprocessor. You run the programs and see the results typed to the screen.

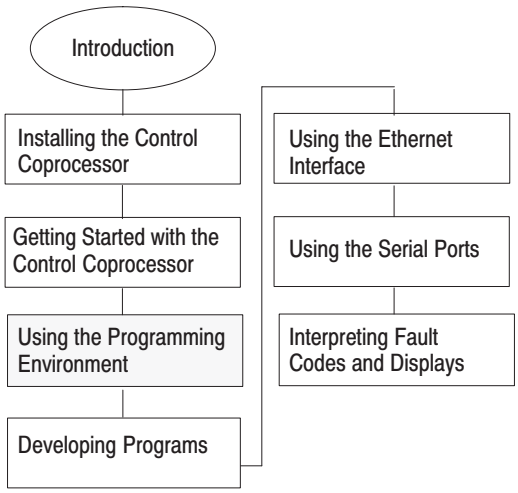
Chapter 4

Using the Programming Environment

Chapter Objectives

This chapter provides an example of creating and compiling a C program using the PCBridge software and the DOS editor; it then shows you how to transfer the program to the control coprocessor. The chapter also provides an example of a BASIC program.

For information on:	See page:
Creating a C test program	4-1
Compiling a C test program	4-2
Sending a binary file to the control coprocessor	4-3
Running a C program on the control coprocessor	4-5
Confirming file passage to the control coprocessor	4-5
Creating a BASIC test program	4-5
Accessing RAM disk	4-6



Create a C Test Program

Create a test C program using the text editor. The default text editor is DTE, a public domain text editor provided for your convenience. Use it to edit small files and PCBridge configuration information. For more information on DTE, view the files DTE.MAN, DTE.DOC, and DTE.HLP in the \PCBRIDGE\EDITOR subdirectory.

Important: You need the 1771-PCB software—installed in Chapter 3—to create C and assembler programs for the control coprocessor.

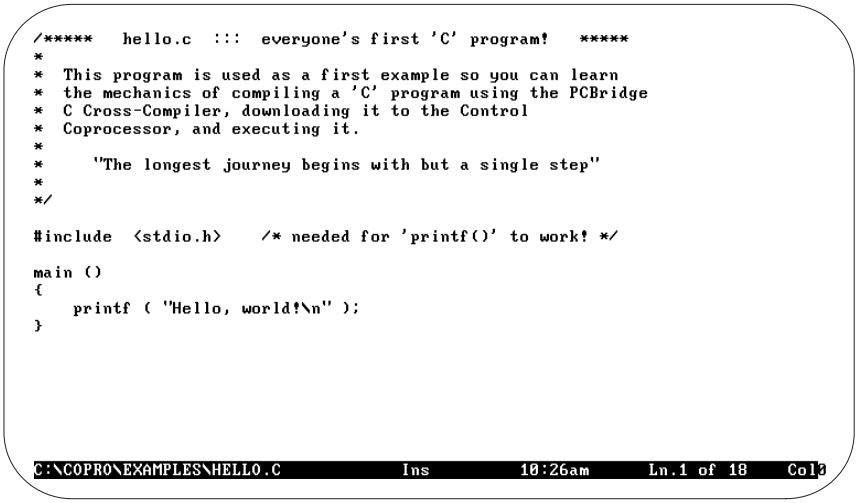
1. If you do not want to use DTE, select `E) Editor Name` on the PCBridge Configuration Options menu and change the text editor.

   Important: The text editor you select must run in 250 Kbytes or less of memory, depending on your system configuration.

2. Select `E) Edit` on the PCBridge main menu to get the text editor.

3. Using your text editor, create the following C test program. See Figure 4.1.

Figure 4.1
C Test Program



This example creates a test file named HELLO.C.

4. Use the exit function on your text editor to return to the PCBridge main menu.

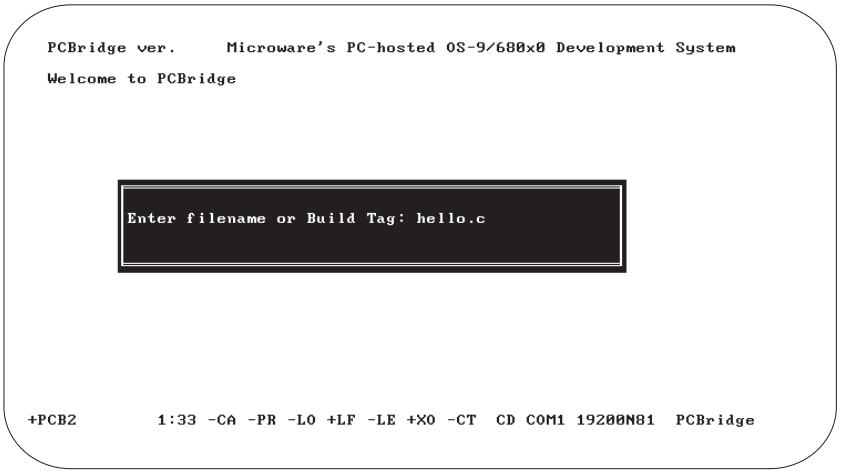
Compile a C Test Program

To compile the C test program:

1. Select `B) Build` on the PCBridge main menu.

2. Enter the name of the test file and press **`[Return]`**. See Figure 4.2.

Figure 4.2
Enter Name of Test File for Compiling



The C cross-compiler function compiles the test file. See Figure 4.3.

Figure 4.3
Cross Compiling C Test File

See the OS-9 C Language User Manual, publication 1771-6.5.104, and Appendix D for more information on setting compiler options.

3. Use the exit function on your text editor to return to the PCBridge main menu.

The result of the build function is a binary, executable file of the program named HELLO.

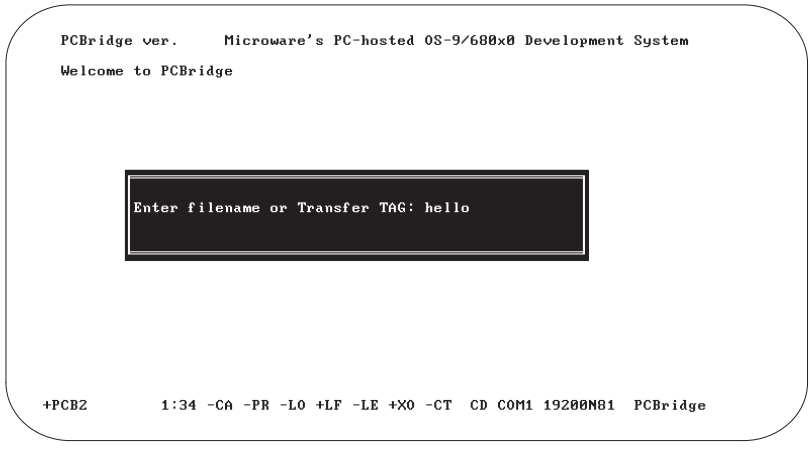
Send a Binary File to the Control Coprocessor

To send the binary file to the control coprocessor:

1. Select `S) Send file to OS-9` on the PCBridge main menu.

2. Enter the name of the compiled file, and press **`[Return]`**.

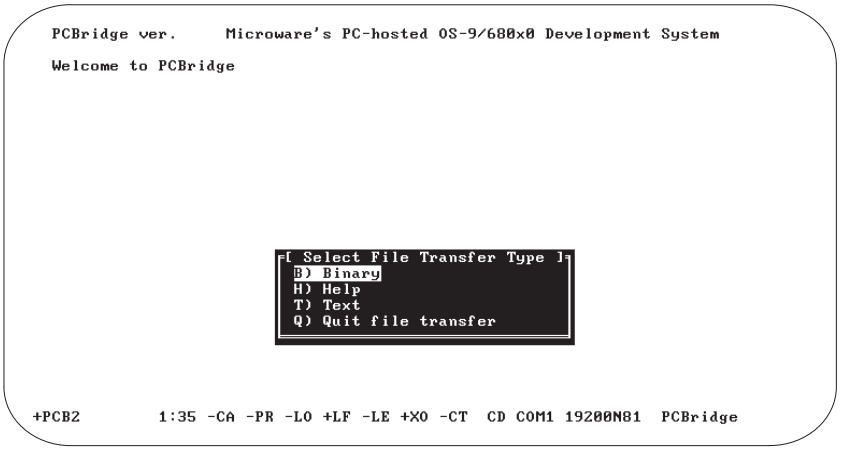
Figure 4.4
Enter Name of Test File to Send to Control Coprocessor



In OS-9, you type the full file name to execute the command. In our example, the full file name is HELLO. The executable file for OS-9 does not have an extension—as compared to an executable DOS file, which has a .COM, .EXE, or .BAT extension.

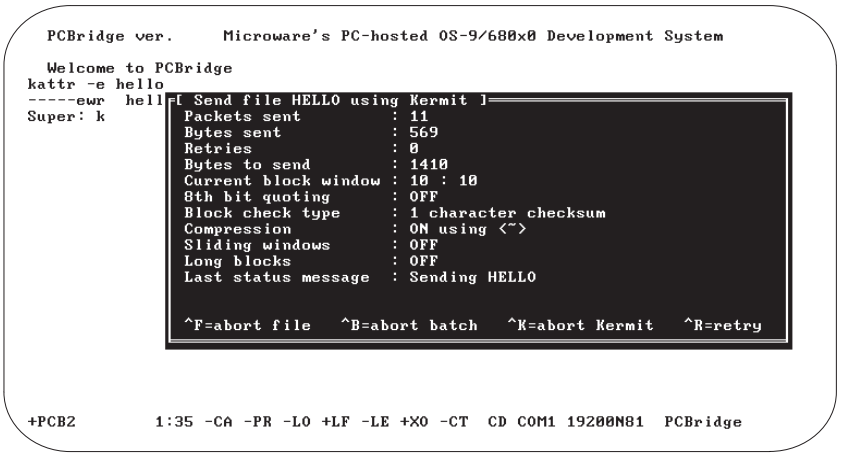
3. Select `B) Binary` on the Select File Transfer Type screen. See Figure 4.5.

Figure 4.5
Select File Transfer Type



The C test file is sent to the control coprocessor via Kermit. You see the screen illustrated in Figure 4.6 being updated while the file is transferred.

Figure 4.6
Kermit Send File



Refer to Appendix D, Using PCBridge, for information on loading memory modules.

Confirm File Passage to the Control Coprocessor

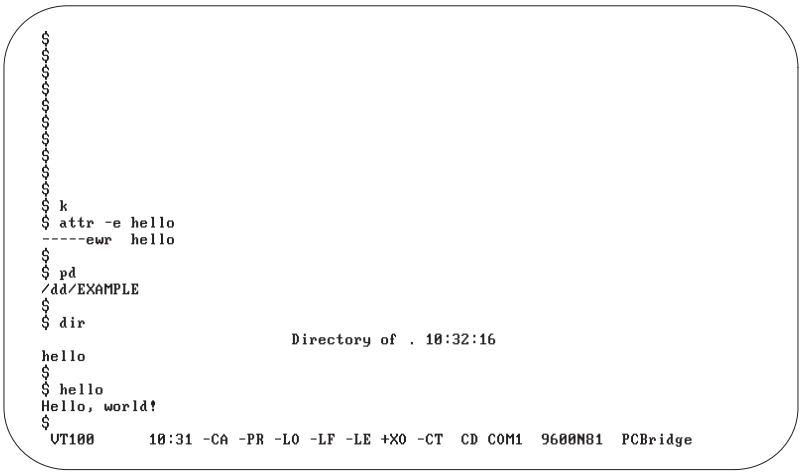
To confirm that the C file is resident in the control coprocessor:

1. Select `O) OS-9 Terminal` on the PCBridge main menu. You get the control-coprocessor OS-9 command-line interface.

   The C file that you previously sent to the control coprocessor should reside in the directory that you last accessed on OS-9.

2. At the $ prompt, type **`dir`** and press **`[Return]`** to verify that the C test file was successfully transferred to the control coprocessor.

Figure 4.7
C Test File on OS-9



3. At the $ prompt, type the name of your executable file—**`hello`**—and press **`[Return]`**. The program executes and prints the `Hello, world!` message.

Create a BASIC Test Program

This section provides a BASIC example program. You can develop and run your BASIC program with either:

- a personal computer with PCBridge software—develop and run your BASIC example program; download it to the control coprocessor via the PCBridge send-ASCII-file function
- an ASCII terminal—use OS-9 to develop and run your BASIC example program

See the OS-9 Operating System User Manual, publication 1771-6.5.102, and the OS-9 BASIC User Manual, publication 1771-6.5.103, for more information on creating and running BASIC programs.

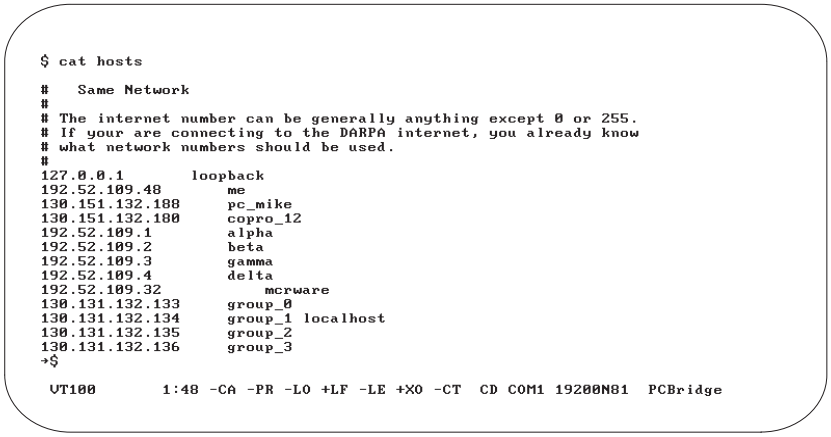
The following example program—HELLO.BAS—is the BASIC version of the C example program.

```
rem    hello.bas
rem  ************************************************************************
rem
rem    This program is used as an example so you can learn the mechanics
rem    of writing a Basic program using the control coprocessor.
rem
rem  ************************************************************************
rem Declare some variables to be used later in the program.
DIM total, x, y: INTEGER
rem Send the ASCII control code "[2J" to clear the screen.
PRINT CHR$(27); "[2J"
rem Print a text string to the screen.
PRINT "Hello, world!"
PRINT
PRINT
rem Now try some math with the variables declared at the beginning.
x = 2
y = 5
total = x + y
PRINT "The results of our math test (x + y) is "; total
PRINT
```

Use an Example Application Program to Access the RAM Disk

This section shows a C program called CAT.C. It reads files from the RAM disk and displays them to the screen; it is a simplified version of the UNIX utility—`cat`—which concatenates files to standard output. Figure 4.8 shows the output from the program CAT.C. The text displayed is the concatenation of two files—HOSTS and HOSTS.EQUIV.

Figure 4.8
Access RAM Disk and Read File to Screen



Example Program to Access RAM Disk

Refer to the following C program (CAT.C) as an example of accessing the control-coprocessor RAM disk. Note the use of standard C library functions—e.g., `fopen()`, `getc()`, and `fclose()`—to access RAM-disk files.

You create the file, compile it, and send it to OS-9 as a binary file. Then, you run the C program on OS-9.

```
/*****   cat.c  :::  copy from files to standard out   *****
*
*  This program is used as an example so you can learn to use the
*  Standard Library functions for processing characters from an
*  input file and writing characters on the standard output,
*  and so you can use a command in a “pipeline” with redirection
*  modifiers (’<’ and ’>’).  Try doing a:
*
*       cat  file1  file2  file3  > outfile
*
*  type of operation to see how “cat” can merge files...
*
*     “What goes around, comes around.”
*/
/*  First, includes and defines...  */
#include  <stdio.h>     /* needed for ’getc()’ and ’putchar()’    */
#include  <errno.h>     /* needed for ’errno’ to work             */
/*  then the function and parameter declarations...  */
main ( argc, argv )
int     argc;
char  **argv;
/*  then the body of executable function statements...  */
{
    /*  private variable declarations  */
    FILE    *infil;     /* file to copy to standard output               */
    int      c,         /* character (or EOF) gotten from input file     */
             i;         /* which command line argument is being processed */
    /*
     *  This command has no option switches.  It simply copies the
     *  input file(s) character by character to standard output.
     *  If only the command itself is specified, it does nothing.
     *  Probably better tell the user such...!
     */
    if ( argc == 1 )       /* no file names on command line */
    {
        fprintf ( stderr, “No files on command line.  Exiting.\n” );
        exit ( 0 );
    }
```

```
    /*
     *  Now, as long as we have files on the command line to process,
     *  open them, read them, and output them to standard output.
     */
    for ( i = 1;  i < argc; i++ )
    {
        /*
         *  Remember, fopen() returns a file pointer.  If the file
         *  pointer returned points to NULL, the file couldn't be
         *  opened.  Exit and tell the user why.
         */
        infil  =  fopen ( argv[i], "r" );
        if ( infil == NULL )
        {
            fprintf ( stderr,
                      "*** cat: unable to open %s.  Aborting\n",
                      argv[i] );
            exit ( errno );
        }
        /*
         *  Now loop to get all the characters in the input file and
         *  put them on the output file ["stdout" for "putchar()"].
         */
        while ( ( c = getc ( infil ) ) != EOF )
            putchar ( c );
        /*
         *  Done with this file.  Close it and get the next one...
         */
        fclose ( infil );
    }
    /*
     *  Done with all the command line arguments, so done with this
     *  program!
     */
    exit ( 0 );
}
```

What to Do Next

When you are familiar with the programming environment, proceed to any of the chapters listed below.

If you want to:	Go to:
Learn to use the Application Program Interface (API) library of routines; you can link these routines to your C and BASIC programs for communication with a programmable controller	Chapter 5
Establish Ethernet communication; see examples of using the Internet Socket Library in C programs	Chapter 6
Establish serial port communication	Chapter 7

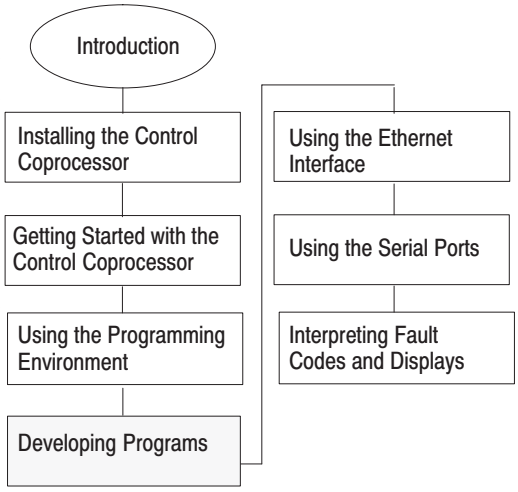
Chapter 5

Developing Programs

Chapter Objectives

This chapter describes the library of commands and executable functions available with the control coprocessor. You will also learn when and how to use them for communication with a programmable controller.

For information on:	See page:
What is the API	5-2
When to use API functions	5-2
Using DTL functions	5-3
Using BPI functions	5-6
Using Message instructions	5-7
Using TAG functions	5-10
Using CC utility functions	5-12
Preparing programs for direct-connect mode	5-14
Preparing programs for standalone mode	5-18



ATTENTION: Control-coprocessor programs that unintentionally write to memory outside their own data space can corrupt memory for other applications or corrupt system memory. This may cause unpredictable control-coprocessor operation, including module reset. In a multi-user environment, a reset naturally affects other users. We strongly recommend that multi-user development be done in an offline or non-critical context.

What Is the Application Program Interface?

The Application Program Interface (API) is a set of library routines used to interface your programs with the control coprocessor. The following are the categories of functions available in the API library.

Table 5.A
API Library Routines

API Function	Definition of Set
DTL	Data-table library (DTL) commands that access the data-table memory of a programmable controller that is directly connected (direct-connect mode) to the control coprocessor
BPI	Control-coprocessor commands accessing the data-table memory of a programmable controller through the backplane interface (BPI)
MSG	Control-coprocessor message (MSG) commands that handle unsolicited Message Instructions from a programmable controller ladder-logic program (direct-connect mode)
TAG	Control-coprocessor commands (TAG) that provide access to the control-coprocessor memory for external devices that are connected via the serial interface(s); ControlView® is an example of such a device that would require access to control-coprocessor memory; TAG also provides access to control-coprocessor memory between OS-9 program modules
CC	Control-coprocessor utility commands that handle functions such as trap initialization, error handling, ASCII displays, etc.

When to Use API Functions

Use Table 5.B to determine which API functions to use for your specific application.

Table 5.B
When to Use API Functions

For this application:	Use this set of API functions;
Access the data table of a PLC-5 programmable controller that is directly connected to the control coprocessor	DTL_
Accomplish discrete or block transfer of data with a programmable controller (either direct-connect or standalone mode)	BPI_
Respond to an unsolicited programmable-controller message	MSG_
Provide access to control-coprocessor memory for interaction among routines running on the control coprocessor and to external devices connected via the serial port(s)	TAG_
Provide access to control-coprocessor memory for devices attached to the serial ports	TAG_
Handle errors generated by API functions	CC_
Initialize the control coprocessor (accomplish first and once only in every program)	CC_

How to Use DTL Functions

Use the DTL library of commands to access real-time data from the data table of a direct-connect PLC-5 programmable controller. The data is transferred between the control coprocessor and the PLC-5 processor via the connector interface between the two devices.

This section defines the available commands. For more details, see Appendix B, Application Program Interface Routines.

Important: You must use the DTL_INIT function to initialize the data-table library before using any data-transfer, data-definition, or chassis-control functions

Configuration Functions

Use configuration functions to initialize the DTL software and establish an internal data-definition table for data items. See Table 5.C.

Table 5.C
DTL Configuration Functions

Function	What It Does	Why You Need It	When You Use It
DTL_INIT	Creates and initializes the data-definition table	You must establish the data-definition table before you call DTL functions	It is required and must be in the DTL function called in your program; you should call it only once per program
DTL_C_DEFINE	Adds a data definition to the data-definition table	For data transfer solicited by a C application program	Required for C code data items
DTL_UNDEF	Deletes a data definition from the data-definition table	To free the data-definition table entry to be reused for another data item	Should be called when a definition is no longer needed
DTL_DEF_AVAIL	Returns the number of data definitions that can be added to the data-definition table	To check that you do not define more data items than the data-definition table can hold	When you want to keep track of how many definitions have been defined at one time

Read/Write Access Functions

Use read/write access functions to exchange data between the directly connected PLC-5 programmable controller and the control coprocessor. See Table 5.D. The read/write functions listed are synchronous to the application program; and they are all control-coprocessor initiated.

The DTL read/write functions are the quickest ways for the control coprocessor to access data in an attached PLC-5 programmable controller. Every read or write interrupts the programmable controller's ladder-program scan for approximately 1 msec, regardless of the length of the transfer; therefore, you should make fewer transfers with greater transfer lengths rather than several small transfers. The amount of time that it takes for the coprocessor to retrieve data and have it available for the application program follows this linear formula:

```
Time (msec) = 1.2 msec + (0.012 msec x number of words)
```

Table 5.D
DTL Read/Write Access Functions

Function	What It Does	Why You Need It	When You Use It
DTL_READ_W	Reads data from the PLC-5 programmable-controller data table to the control-coprocessor memory	To perform a read of a PLC-5 programmable-controller data table	When you want to receive data from the PLC-5 programmable-controller data table
DTL_READ_W_IDX	Reads any elements of a file, one element at a time, from the PLC-5 programmable controller to the control-coprocessor memory using only one data definition	To perform an indexed read of a PLC-5 programmable-controller file	When you want to receive any elements of a file from the PLC-5 programmable-controller data table using one data definition
DTL_WRITE_W	Writes data from the control-coprocessor memory to the PLC-5 programmable-controller data table	To perform a write to a PLC-5 programmable-controller data table	When you want to write data to a PLC-5 programmable-controller data table
DTL_WRITE_W_IDX	Writes any elements of a file, one element at a time from the control-coprocessor memory to the PLC-5 programmable controller using only one data definition	To perform an indexed write to a PLC-5 programmable-controller file	When you want to write any elements of a file to a PLC-5 programmable-controller data table using one data definition
DTL_RMW_W	Initiates an operation that: • reads a data element • modifies some of the bits • then writes it back	To perform a read/modify/write to a PLC-5 programmable-data element	When you want the application program to read/modify/write an element of the PLC-5 programmable-controller data table
DTL_RMW_W_IDX	Initiates an operation that reads a data element of the PLC-5 processor, modifies some of the bits based on mask values, then writes the data element back	To perform an indexed read/modify/write to a PLC-5 programmable-controller file	When you want the application program to read/modify/write any elements of a PLC-5 programmable-controller file using only one data definition

Conversion Functions

Conversion functions convert data from one format to another. When you specify an application data type in the definition of a data item, the read, write, and receive functions automatically convert the data from the format in the PLC-5 programmable controller to proper format for the control coprocessor. The data types are as follows:

PLC Data Types	Control-Coprocessor Data Types	
signed word IEEE float	raw byte ubyte word uword	long ulong float double

See Table 5.E for DTL conversion functions.

Table 5.E
DTL Conversion Functions

Function	What It Converts
DTL_GET_WORD	2-byte array to host data-type word
DTL_GET_FLT	Raw 32-bit IEEE float data, in 4-byte array, to host type float
DTL_GET_3BCD	A 3-digit BCD value stored in a 2-byte array to a control-coprocessor unsigned①
DTL_GET_4BCD	A 4-digit BCD value stored in a 2-byte array to a control-coprocessor unsigned①
DTL_PUT_WORD	Control-coprocessor unsigned to a 2-byte array①
DTL_PUT_FLT	Control-coprocessor float to a 4-byte array in IEEE 32-bit binary format
DTL_PUT_3BCD	Control-coprocessor unsigned to 2-byte, 3-digit BCD value①
DTL_PUT_4BCD	Control-coprocessor unsigned to 2-byte, 4-digit BCD value①

① Unsigned is the same as unsigned integer or unsigned long

Control-Coprocessor Memory Functions

Use DTL_SIZE and DTL_TYPE functions to determine the size and location of control-coprocessor memory required to store the contents of the data item in the control-coprocessor format. See Table 5.F.

Table 5.F
DTL Memory Functions

Function	What It Does
DTL_SIZE	Determines the amount of control-coprocessor memory necessary to store the defined block of data
DTL_TYPE	Gets the data type of the defined block of data specified in DTL_C_DEFINE

Utility Function

Use the DTL_CLOCK function to synchronize the control-coprocessor date and time with that of the PLC-5 programmable controller. The control-coprocessor time is synchronized within 1 second of the PLC-5 programmable-controller clock. This is a one-time-only synchronization. The application can maintain synchronization by calling DTL_CLOCK at regular intervals.

How to Use BPI Functions

The control coprocessor communicates with a standalone-mode programmable controller using backplane-interface (BPI) functions. The communication is via the 1771 I/O chassis backplane. You can also use the BPI functions when you have a PLC-5 programmable controller directly connected to the control coprocessor.

For backplane communication, the control coprocessor appears to the programmable controller as a 16-bit, bidirectional I/O module. The control coprocessor can perform both discrete- and block-data transfers.

Important: The only bits available for use by the application program are the upper 8 bits (10-17). The lower 8 bits (0-7) are reserved for block transfer, even if there are no block transfers programmed to the control coprocessor.

You must prepare a control-logic program in the programmable controller to initiate block transfer and/or discrete reads and writes with the control coprocessor. See page 5-21 for an example of a control-logic program.

ATTENTION: The control coprocessor will not communicate via discrete or block transfer in any chassis (remote or local) set for 2-slot addressing; however, 1-slot and 1/2-slot addressing are valid configurations for a chassis that contains a control coprocessor communicating via discrete or block transfer with a PLC processor.

Update Discrete Data

Use BPI_DISCRETE to get the updated output-image word from the PLC-5 programmable controller or send the input-image word to the controller. The function determines whether it is an input or an output word.

Accomplish Block-Transfer Reads and Writes

Use BPI_WRITE and BPI_READ routines to allow PLC-5 programmable-controller reads and writes of block data across the backplane interface.

Function	What It Does
BPI_WRITE	This routine interfaces with a synchronous block-transfer read from a programmable controller
BPI_READ	This routine interfaces with a synchronous block-transfer write from a programmable controller

How to Use Message Instructions

The control coprocessor can receive unsolicited messages from the PLC-5 programmable controller. Two types of messages are supported:

- read data (word-range) from the control coprocessor
- write data (word-range) to the control coprocessor

The control coprocessor supports up to 32 unsolicited messages. The control-coprocessor message numbers are 0-31 (ASCII). Use the control-coprocessor MSG library of commands with a directly connected PLC-5 programmable controller.

PLC-5 Programmable-Controller MSG Instruction

A PLC-5 programmable controller uses the message (MSG) instruction for unsolicited communication with the control coprocessor. You program the MSG instruction in the ladder logic of the PLC-5 programmable controller. This PLC-5 programmable-controller communication with the control coprocessor is through the direct-connect mode side connector (Port 3A).

You specify a control-block address when you first enter the MSG instruction. The programming terminal then automatically displays a data-entry screen, where you enter instruction parameters that are stored at the control-block address. You can also use the data-monitor screen to edit selected parameters of the MSG instruction.

See the PLC-5 Programming Software Instruction Set Reference, publication 6200-6.4.11, for more information on the message instruction.

PLC-5 Ladder-Logic Program
Enter the information in the PLC-5 programmable-controller message control block. See Figure 5.1. Use only the choices listed:

- communication commands—PLC-3 word-range read or PLC-3 word-range write
- destination data-table address—“00” through “31”
- port number—3A

Important: On the 6200 Series Programming Software data-entry screen, specify 3A for the communication port number. **You must use the MG data type for the control block if you want to set the port number.**

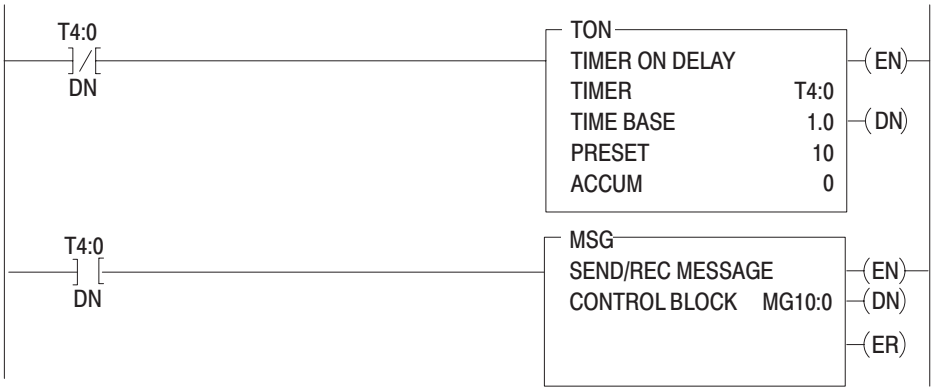
Figure 5.1
MSG Instruction Data-Entry Screen (MG Control Block)

```
            MESSAGE INSTRUCTION DATA ENTRY FOR CONTROL BLOCK MG10:10

                Communication Command              PLC-3 Word Range Write
               PLC-5 Data Table Address:          N7:3
               Size in Elements:                  1
               Local/Remote:                      LOCAL
                  Remote Station:                 N/A
                  Link ID:                        N/A
                  Remote Link Type:               N/A
               Local Node Address:                00
               Destination Data Table Address:    "30"
               Port Number                          3A

  BLOCK SIZE IS 56



Press a key to change a parameter or <ENTER> to accept parameters.

Program    Forces:None      Edits:None                    PLC-5/40 File DRILL1
 Read/   PLC-5  Size in Local/  Remote   Link   Remote   Local  Destin   Port
 Write  Address Elemnts Remote  Station   ID     Link    Node  Address   Number
  F1      F2      F3      F4      F5      F6      F7      F8      F9      F10
```

Sample PLC-5 Ladder-Logic Program

Use the following sample ladder-logic program as a guide when you prepare programs for your application. This example triggers or initiates a message instruction every 10 seconds.

Figure 5.2
Example Ladder-Logic Program for PLC-5 Programmable Controller



Control-Coprocessor Message Functions

The control coprocessor has message (MSG) functions that process unsolicited messages from the PLC-5 programmable controller. The control coprocessor supports both synchronous and asynchronous message functions.

This section defines the MSG functions that process unsolicited messages from a PLC-5 programmable controller. See Appendix B, Application Program Interface Routines, for more information.

Read/Write MSG Functions
Use read/write MSG functions to process unsolicited MSG instructions from a PLC-5 ladder-logic program. See Table 5.G.

Table 5.G
Control-Coprocessor MSG Read/Write Processing Functions

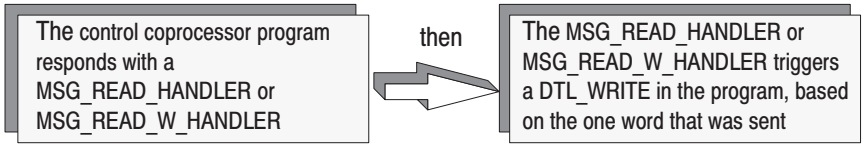
Function	What It Does	Why You Need It	When You Use It
MSG_READ_W_HANDLER	Processes an unsolicited PLC-5 MSG read instruction	To perform synchronous processing of an unsolicited PLC-5 MSG read instruction	When you want to transfer read data to the PLC-5 programmable controller before the next step of the application program is executed
MSG_READ_HANDLER	Initiates processing of an unsolicited PLC-5 MSG read instruction	To perform asynchronous processing of an unsolicited PLC-5 MSG read instruction	When you want to transfer read data to the PLC-5 programmable controller but return control to the application program before the message is executed (also, see MSG_WAIT function)
MSG_WRITE_W_HANDLER	Processes an unsolicited PLC-5 MSG write instruction	To perform synchronous processing of an unsolicited PLC-5 MSG write instruction	When you want to receive write data from the PLC-5 programmable controller before the next step of the application program is executed
MSG_WRITE_HANDLER	Initiates processing of an unsolicited PLC-5 MSG write instruction	To perform asynchronous processing of an unsolicited PLC-5 MSG write instruction	When you want to receive write data from the PLC-5 programmable controller but return control to the application program before the message is executed (also, see MSG_WAIT function)

When a PLC-5 programmable controller generates a Message READ instruction, a corresponding MSG_READ_HANDLER or MSG_READ_W_HANDLER function must be used by the control coprocessor to handle the request. When a PLC-5 programmable controller generates a Message WRITE instruction, a corresponding MSG_WRITE_HANDLER or MSG_WRITE_W_HANDLER function must be used by the control coprocessor to handle the request.

When a PLC-5 programmable controller initiates an unsolicited read/write MSG instruction, use read/write MSG functions in the control coprocessor for **transferring small amounts of data** between the programmable controller and the control coprocessor. The maximum size that you can specify for the buffer in the read/write MSG function is 240 bytes for a read and 234 bytes for a write.

For **transferring larger amounts of data** between the programmable controller and the control coprocessor, use a read/write MSG function in conjunction with a read/write DTL function in your control-coprocessor program. You can transfer up to 1,000 words between the control coprocessor and a PLC-5 programmable controller using the read/write DTL function.

For example, a PLC-5 programmable controller initiates an unsolicited READ MSG instruction to the control coprocessor. The READ MSG instruction transfers one word of data. The order of events is:



Note: The DTL_WRITE function writes data from the control coprocessor to the PLC-5 programmable controller.

Check Status of Asynchronous MSG Functions (MSG_WAIT)
When your application program uses asynchronous MSG read/write functions, you must include at least one MSG_WAIT in your program. MSG_WAIT checks for the completion of any combination of pending read/write message number and updates the message number in a read and write mask.

Mask Functions
Use the MSG_ZERO_MASK function to zero all the bits in a mask from previous operations when you use a MSG_WAIT function in your program. All the mask functions are used with the MSG_WAIT function.

Table 5.H
Mask Functions for Unsolicited, Asynchronous Read and Write Functions

Function	What It Does
MSG_CLEAR_MASK	Clears the bits in the message read/write masks associated with a specified message number
MSG_SET_MASK	Sets the bits in the message read/write masks associated with a specified message number
MSG_TST_MASK	Tests the bits in the message read/write masks associated with a specified message number
MSG_ZERO_MASK	Used to zero all the bits in a specified message number

Clear Pending Messages
See page 5-13 for information on using the CC_MKILL utility to clear pending messages from the message handler.

How to Use TAG Functions

TAG functions provide a means for the user to specify access to control-coprocessor memory. Use TAG functions to access memory for:

- external intelligent devices—i.e., ControlView connected to a serial port
- multiple processes interacting on OS-9

You can configure the size of the TAG table using the CC_CFG utility. The default size allows the creation of 1024 TAGs.

Important: For the 1771-DMC1 and -DMC4 modules, the default size allows you to create 1024 TAGs; the -DMC module default size is zero.

Use the following sections to select the appropriate TAG function for your application. See Appendix B, Application Program Interface Routines, for the following information on these TAG functions: description, required parameters, condition values, and a C program example.

TAG-Table Configuration Functions

Use these TAG configuration functions to establish a TAG table for TAG functions:

Table 5.I
TAG-Table Configuration Functions

Function	What It Does
TAG_DEFINE	Places a TAG name entry into the TAG table
TAG_UNDEF	Removes a TAG name or TAG names from the TAG table
TAG_DEF_AVAIL	Determines the number of TAG definitions available in the TAG table
TAG_GLOBAL_UNDEF	Removes a TAG or TAGs from the TAG table defined by any process
TAG_LINK	Gets a handle (offset) for a TAG table entry

Read and Write Functions

Use these TAG functions to read and write to coprocessor memory:

Table 5.J
TAG Read and Write Functions

Function	What It Does
TAG_READ	Reads data from a tagged memory area
TAG_READ_W	Reads data from a tagged memory area after the tag has been written by TAG_WRITE_W
TAG_WRITE	Writes data to a tagged memory area
TAG_WRITE_W	Writes data to a tagged memory area, returns only after the tag has been read by TAG_READ_W

Lock/Unlock Functions

Use the TAG_LOCK and TAG_UNLOCK functions as a pair in your program. The TAG_LOCK function protects against concurrent access to the tagged area of control-coprocessor memory. The TAG_UNLOCK function is an unlock to the TAG locked by the TAG_LOCK function.

Important: Failure to use a TAG_UNLOCK function to complement a TAG_LOCK function in a program may cause the system to hang-up.

How to Use CC Utility Functions

This section covers CC utility functions of the control coprocessor such as:

- initialization
- error handling
- ASCII display interface functions
- synchronizing a control-coprocessor calling task to a PLC-5 programmable-controller ladder-logic program scan

Initialize Control-Coprocessor Function

Use the CC_INIT function to initialize the control coprocessor.

Important: The **CC_INIT function must be called** before you can use any other API library function. Call the CC_INIT function first and once only in your program.

Control-Coprocessor Error Functions

Use control-coprocessor error functions for error messages related to error numbers. See Table 5.K.

Table 5.K
Error Functions

Function	What It Does
CC_ERROR	Provides a pointer to an error message for a corresponding error number (for all API functions); typically used in a C routine
CC_ERRSTR	Copies the error message to a corresponding error number to the user local buffer (for all API functions); typically used in a BASIC procedure

Control-Coprocessor ASCII Display Functions

Use the ASCII display functions to show control-coprocessor status information on the optional serial expander module display. See Table 5.L.

Table 5.L
ASCII Display Functions

Function	What It Does
CC_DISPLAY_STR	Displays a 4-character string on the ASCII display
CC_GET_DISPLAY_STR	Returns the value of the current ASCII display to user buffers
CC_DISPLAY_HEX	Displays a 3-character hexadecimal value on the ASCII display
CC_DISPLAY_EHEX①	Displays a 4-character hexadecimal value on the display
CC_DISPLAY_DEC①	Displays a 4-character decimal value on the ASCII display

① You must know whether the fault displayed is hexadecimal or decimal when you use the 4-character display. For example, if the ASCII display is 1234, it could be either a hexadecimal or a decimal representation.

Synchronization Function

Use the CC_PLC_SYNC function to synchronize the control-coprocessor calling task to the PLC-5 programmable-controller ladder-program scan. This function automatically puts the current application to sleep until the start of the next PLC-5 program scan.

Because of the multi-tasking feature of OS-9, it is most effective to synchronize only one priority task to the PLC-5 programmable-controller ladder-program scan.

Status Function

Use the CC_PLC_STATUS function to get the current status of the PLC-5 programmable controller.

Use the CC_STATUS and CC_EXPANDED_STATUS functions to get the current status of the control coprocessor.

Clear Message-Handler Function

Use the CC_MKILL utility to clear pending message handlers so they are available for use by other applications. For example, messages can still be pending for an aborted or terminated application that was performing message functions. These pending messages can cause the message handler to be pending indefinitely.

Syntax for the CC_MKILL utility is:

```
cc_mkill {<opts>}
Function:  Kill PLC Message Entry
Options:   -r=<num>     Kill Pending Read Message (0-31,[*])
           -w=<num>     Kill Pending Write Message (0-31,[*])
           -a=*         Kill All Read/Write Messages
```

Using this utility, you can clear one entry at a time or you can clear all message handlers. The following example clears the read handler for message 13:

```
$ cc_mkill -r=13
```

The following example clears all message handlers:

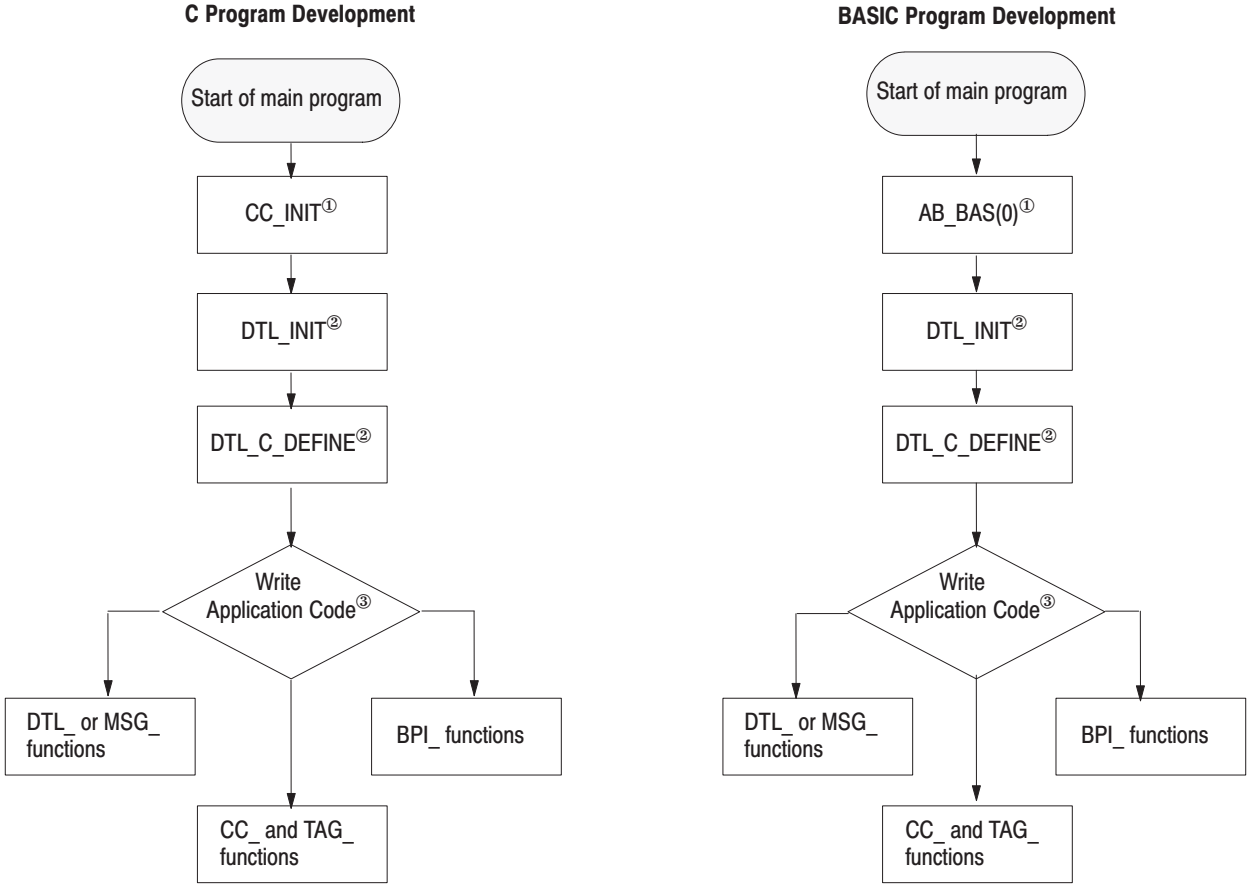
```
$ cc_mkill -a=*
```

Prepare Programs for Direct-Connect Mode

In direct-connect mode, the control coprocessor can communicate directly with a PLC-5 programmable controller using DTL and MSG functions. Also, in direct-connect mode, you can use BPI functions for backplane communication with a programmable controller. See page 5-18 for more information on programs using BPI functions.

ATTENTION: The control coprocessor does not incorporate hardware memory protection between processes. It is the user's responsibility to ensure that programs do not overwrite memory used by other programs or by the operating system. This could result in unpredictable system operation.

Direct-Connect Program Requirements and Flow



① Function must be included first and once in each program. Note that for BASIC programs, CC_INIT is accessed using AB_BAS(0).
② Only necessary when using DTL_ functions.
③ The multi-tasking operating system can perform application processes that include combinations of the API functions.

Link API Functions to Programs

In C and assembler programs, ABLIB.L provides the interface (link) to the library of control-coprocessor API functions.

In BASIC programs, AB_BAS provides the interface (link) to the library of control-coprocessor API functions.

In C, BASIC, and assembler programs, use the CC_INIT function—use AB_BAS(0) for BASIC programs—to initialize the control coprocessor. The CC_INIT function must be called first and once only in every application program that uses CC_ or DTL_ functions.

Sample C Program

The following is a C programming example. This example program uses DTL_WRITE_W and DTL_READ_W functions to write data to a PLC-5 integer file, read the data, and display the data on the ASCII fault display—on the optional serial expander module. The program continues to write and read, incrementing the fault display value, while continuously checking for errors.

Important: The CC_INIT function must be used in the following program. Call the CC_INIT function first and once only in your program.

Use CC_INIT first and once in every program →

```
/*********************************************************************
*  DTL_W_R.C – This program uses both the DTL_WRITE_W and DTL_READ_W
*  functions.  It writes a single word to the PLC5's N7:0 file and then
*  reads it back.  The copro then copies the data to the 4-digit display
*  on the expander module.  It will do this forever until the program
*  is terminated with a CTRL-E or a kill command from the OS-9 command line.
*********************************************************************/
#include <stdio.h>
#include <copro.h>
char * CC_ERROR();
main()
{
   register unsigned ret_val;              /* private variable declarations */
   unsigned iostat;
   unsigned id_n7;
   unsigned short buffer[1];
   CC_INIT();                                 /* initialize the coprocessor */
   ret_val = DTL_INIT (1);                             /* just 1 definition */
if (ret_val != DTL_SUCCESS)
      {
         print_error (ret_val);                          /* print error... */
         exit(-1);                                             /* and exit */
      }
   ret_val = DTL_C_DEFINE (&id_n7, "N7:0,1,WORD,MODIFY");  /* init for n7:0 */
   if (ret_val != DTL_SUCCESS)
      {
         print_error (ret_val);
         exit (-1);
      }
   buffer[0] = 0;                                    /* initialize data word */
   while (1)                                        /* let's do this forever */
      {
         ret_val = DTL_WRITE_W (id_n7, buffer, &iostat);   /* write to PLC5 */
         if (ret_val != DTL_SUCCESS)                        /* check ret_val */
            break;
         ret_val = DTL_READ_W (id_n7, buffer, &iostat);   /* read back data */
         if (ret_val != DTL_SUCCESS)                        /* check ret_val */
            break;
         CC_DISPLAY_DEC (buffer[0]);                   /* display buffer[0] */
         buffer[0] += 1;                               /* keep incrementing */
         if (buffer[0] == 9999)             /* we are past the limit of the */
            buffer[0] = 0;                                /* 4 digit display */
         tsleep (10);                    /* give us time to see the display */
      }
   print_error (ret_val);           /* oops, we got an error - print it out */
   print_status (iostat);                            /* print status also */
   exit (-3);                                          /* get out of here */
}
print_error (err)                                    /* process error code */
int err;
{
   char *errptr;
   errptr = CC_ERROR (err);                         /* get pointer to string */
   printf ("\n Return Value = %4d - %s \n", err, errptr);       /* print it */
}
print_status (stat)                                   /* process status code */
int stat;
{
   char *errptr;
   errptr = CC_ERROR (stat);                        /* get pointer to string */
   printf ("\n Status Value = %4d - %s \n", stat, errptr);      /* print it */
}
```

Sample BASIC Program

The following is a BASIC programming example that illustrates the interface to various API functions. The program uses CC_ERRSTR to copy the status of the various functions and display the string to the terminal—i.e., CC_INIT, DTL_INIT, DTL_CLOCK, and DTL_READ_W.

```
rem ********************************************************************************
rem *  DEMO.BAS - This basic program demonstrates a few AB API functions          *
    ********************************************************************************
procedure DEMO
   DIM ret_val           : INTEGER
   DIM name_id           : INTEGER
   DIM iostat            : INTEGER
   DIM avail             : INTEGER
   DIM rcvbuff           : INTEGER
   DIM buffer            : STRING[81]
rem * CC_INIT - This call must be made before any other API functions are called
   RUN AB_BAS (0)
rem * CC_DISPLAY_STR - Display the string -AB- on expander module
   RUN AB_BAS (102,ret_val,"-AB-")
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * DTL_INIT - Initialize DTL for 4 definitions
   RUN AB_BAS (1,ret_val,4)
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * DTL_CLOCK - synchronize our clock with the PLC-5
   RUN AB_BAS (18,ret_val)
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * DTL_C_DEFINE - Define a data element
   RUN AB_BAS (2,ret_val,ADDR(name_id),"n7:0,1,long,MODIFY")
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * DTL_DEF_AVAIL - How many are available now? (4 - 1 = ?)
  RUN AB_BAS (4,ret_val,ADDR(avail))
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * Print how many definitions are available now
   print avail
rem * DTL_READ_W - Read form N7:0 1 word into rcvbuff
   RUN AB_BAS (5,ret_val,name_id,ADDR(rcvbuff),ADDR(iostat))
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
rem * Print iostat
   print iostat
rem * Print the read data
   print rcvbuff
rem * CC_DISPLAY_EHEX - Display read data to the expander display
   RUN AB_BAS (105,ret_val,rcvbuff)
rem * CC_ERRSTR - Get the string for the ret_val - display on terminal
   RUN AB_BAS (101,ret_val,ret_val,ADDR(buffer))
   print buffer
   end
```

CC_INIT function is used; called by RUN AB_BAS(0)

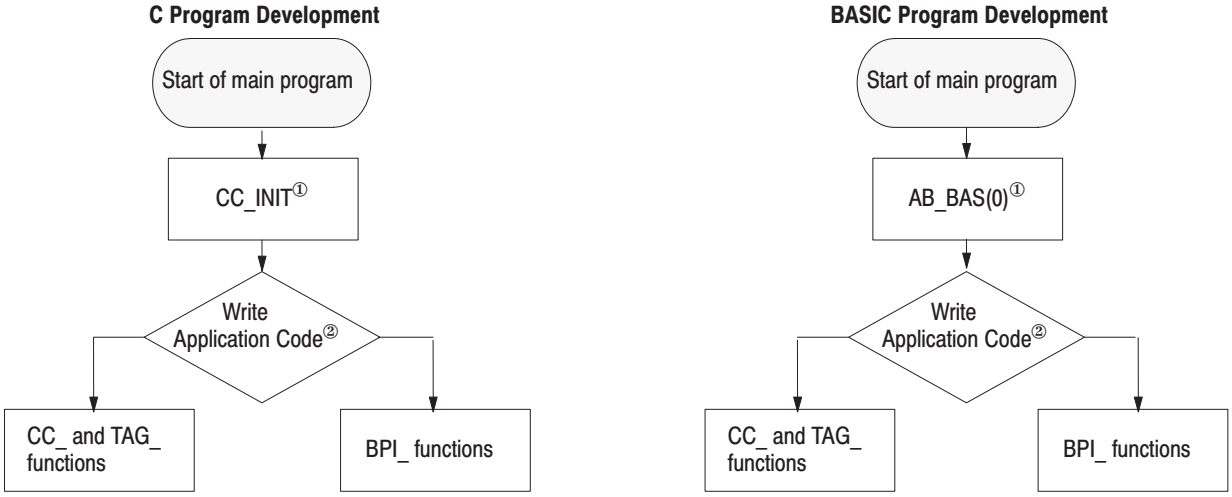
This is the BASIC function number. See Appendix B.

Prepare Programs for Standalone Mode

In standalone mode, use BPI_ functions to communicate with a programmable controller.

ATTENTION: The control coprocessor does not incorporate hardware memory protection between processes. It is the user's responsibility to ensure that programs do not overwrite memory used by other programs or by the operating system. This could result in unpredictable system operation.

Standalone Program Requirements and Flow



① Function must be included first and once in each program. Note that for BASIC programs, CC_INIT is accessed using AB_BAS(0).
② The multi-tasking operating system can perform application processes that include combinations of the BPI_, CC_, and TAG_ functions.

Link API Functions to Programs

In C and assembler programs, ABLIB.L provides the interface (link) to the library of control-coprocessor API functions.

In BASIC programs, AB_BAS provides the interface (link) to the library of control-coprocessor API functions.

In C, BASIC, and assembler programs, use the CC_INIT function—use AB_BAS(0) for BASIC programs—to initialize the control coprocessor. The CC_INIT function must be called first and once only in any program that uses CC_ or DTL_ functions.

Sample C Program

The following is a C programming example. It uses both BPI_WRITE and BPI_READ functions to trigger programmable-controller block-transfer writes and reads. See page 5-21 for more information.

Important: The CC_INIT function is used in the following program. Call the CC_INIT function first and once only in your program.

```
/*********************************************************************
*  bt_w_r.c
*  This program uses both the BPI_WRITE and BPI_READ functions.  It
*  continuously triggers the PLC to do block transfer writes to the
*  coprocessor.  The coprocessor write the data to the user screen
*  and then copies the first five words to the out_buffer that will be
*  sent back to the PLC as part of this loopback test.
*********************************************************************/
#include <stdio.h>
#include <copro.h>
#define CLEAR_SCREEN() printf("\33[2J")                  /* Clear screen macro */
#define MOVE(x,y)      printf("\33[%d;%dH", y, x)         /* Move cursor macro */
#define TIMEOUT 4
#define R_LENGTH 10
#define W_LENGTH 5
#define R_TRIG 0x0100
#define W_TRIG 0x0200
main()
{
   register unsigned ret_val;                /* private variable declarations */
   int x, y = 0;
   unsigned short in_buffer[10];
   unsigned short out_buffer[5];
   for (x=0; x < 10; x++)                              /* Initialize in buffer */
      in_buffer[x] = 0;
   x = 0;                                             /* Reinitialize for later */
   CC_INIT();                                      /* initialize the coprocessor */
   CLEAR_SCREEN();
   while (1){
      /*  Trigger the BTW from the PLC with this BPI_READ function */
      ret_val = BPI_READ (R_LENGTH, in_buffer, TIMEOUT, R_TRIG);
      if (ret_val != DTL_SUCCESS){
        print_error (ret_val);
        exit(10);                                       /* Print error and exit */
      }
for (x=0; x < 5; x++)                                 /* Copy first 5 words of */
         out_buffer[x] = in_buffer[x];               /* input to output buffer */
         /* Print the results of the block transfer continuously on the screen */
      MOVE(0,2);                                  /* Move cursor to top of screen */
      for (x=0; x < 10; x++)
         printf ("Word %d of the PLC BTW = %d \n", x, in_buffer[x]);
      printf ("\n\nFree running timer showing communication activity   %d\n", y++);
      /* Now trigger the PLC to do a BTR with this BPI_WRITE function */
      ret_val = BPI_WRITE (W_LENGTH, out_buffer, TIMEOUT, W_TRIG);
      if (ret_val != DTL_SUCCESS){
         print_error (ret_val);
         exit(20);                                      /* Print error and exit */
      }
   }                                                    /* End of while(1) loop */
}                                                               /* End of main */
print_error (err)                                     /* process error code */
int err;
{
   char *errptr;
   errptr = (char *) CC_ERROR (err);                 /* get pointer to string */
   printf ("\n Return Value = %4d - %s \n", err, errptr);          /* print it */
   printf ("\n\nIf you got a time out error, check that you are in RUN mode\n");
}
```

Use CC_INIT first and once in every program

Sample BASIC Program

The following is a BASIC programming example. It uses both BPI_WRITE and BPI_READ functions to trigger programmable-controller block-transfer writes and reads.

```
rem Sample BPI Basic Program
dim retval, rdata(10),wdata(5),loopcnt,timeout,trigmask,i:integer
dim Wlength,rlength:byte
    loopcnt=0
rem timeout is number of seconds BPI_BTR or BPI_BTW will be attempted by CO_PRO
    timeout=4
rem rlength is BTW length in plc; when CO-PRO does a BPI_BTR, plc does a BTW
    rlength=10
rem wlength is BTR length in plc; when CO-PRO does a BPI_BTW, plc does a BTR
    wlength=5
rem trigmask is value written to input image table; can be used to trigger BTs
    trigmask=0
rem initialze the BPI_BTW data
    FOR i=1 to 5
        wdata(i)=0
    NEXT i
rem initialize coprocessor with CC_INIT, which is AB_BASIC function number 0.
     RUN AB_BAS(0)
rem do a clear screen
  PRINT CHR$(27);"[2J"
1 rem line #1 used for got to
rem do a logical or of trig mask to set bit 10
     trigmask=LOR(256,trigmask)
rem BPI_BTR
      RUN AB_BAS(34,retval,rlength,addr(rdata(1)),timeout,trigmask)
      IF retval<>0 THEN
           PRINT CHR$(27);"[15,0H"
           PRINT "BTR retval=";retval
rem return the first 5 words of the BPI_BTR in the BPI_BTW
      ELSE
           FOR i = 1 TO 5
               wdata(i)=rdata(i)
           NEXT i
rem cursor home
           PRINT CHR$(27);"[H"
           FOR i= 1 TO 10
               PRINT "WORD ";i;" OF PLC BTW = ";rdata(i)
           NEXT i
rem erase to end of screen
           PRINT CHR$(27);"[J"
     ENDIF
rem reset bit 10
      trigmask=LAND(-257,trigmask)
rem just a screen activity indicator
PRINT CHR$(27);"[13,0H SCREEN REFRESH INDICATOR ";loopcnt
        loopcnt=loopcnt+1
rem do a logical or of trig mask to set bit 11
     trigmask=LOR(512,trigmask)
rem BPI_BTW
      RUN AB_BAS(33,retval,wlength,addr(wdata(1)),timeout,trigmask)
      IF retval<>0 THEN
           PRINT CHR$(27);"[16,0H"
           PRINT "BTW retval=";retval
      ENDIF
rem reset bit 11
      trigmask=LAND(-513,trigmask)
GOTO 1
```

CC_INIT function is used; called by RUN AB_BAS(0)

This is the BASIC function number. See Appendix B.

Sample Control-Logic Program

The following is a control-logic programming example. This control-logic program initiates a block-transfer write and a read when triggered by the control coprocessor. See the sample C—on page 5-18—and BASIC— on page 5-20—control-coprocessor programs.

```
Rung 2:0
This block-transfer write will be triggered when bit 10 of the input image is
set by the control coprocessor.  The coprocessor triggers this bit using the
trigger mask within the BPI_READ function.  The second conditional bit is the
enable bit of the block-transfer control block.  This ensures that the rung
will be toggled false to true every time the block transfer completes.  The
coprocessor is located in rack 0, group 2 of the chassis.

| I:002 N10:0                                  +BTW--------------------+      |
+--] [---]/[----------------------------------+BLOCK TRNSFR WRITE     +-(EN)-+
|    10    15                                  |Rack                 00|      |
|                                              |Group                 2+-(DN) |
|                                              |Module                0|      |
|                                              |Control Block     N10:0+-(ER) |
|                                              |Data file        N10:10|      |
|                                              |Length               10|      |
|                                              |Continuous            N|      |
|                                              +----------------------+      |


Rung 2:1
This block-transfer read will be triggered when bit 11 of the input image is
set by the control coprocessor.  The coprocessor triggers this bit using the
trigger mask within the BPI_WRITE function.  The second conditional bit is the
enable bit of the block-transfer control block.  This ensures that the rung
will be toggled false-to-true every time the block transfer completes.

| I:002 N11:0                                  +BTR--------------------+      |
+--] [---]/[----------------------------------+BLOCK TRNSFR READ      +-(EN)-+
|    11    15                                  |Rack                 00|      |
|                                              |Group                 2+-(DN) |
|                                              |Module                0|      |
|                                              |Control Block     N11:0+-(ER) |
|                                              |Data file        N11:10|      |
|                                              |Length                5|      |
|                                              |Continuous            N|      |
|                                              +----------------------+      |
Rung 2:2
|                                                                            |
+------------------------------[END OF FILE]-------------------------------+
|                                                                            |
NO MORE FILES
```

Chapter 6

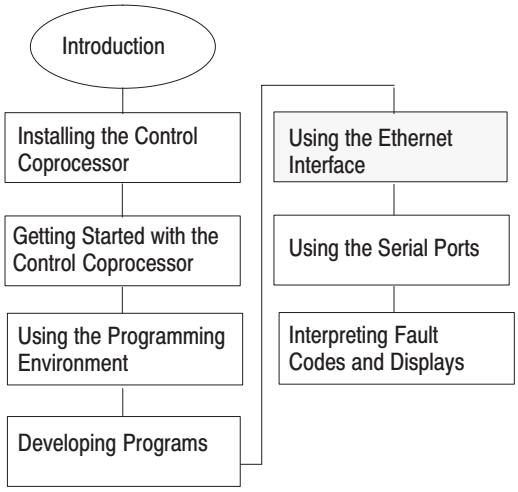
Using the Ethernet Interface

Chapter Objectives

This chapter provides an overview of the Ethernet local area network capability of the control coprocessor. It provides information on how to:

- connect to the network using a transceiver (Medium Access Unit)
- configure the Ethernet port
- send/receive communication using the FTP and TELNET utilities
- prepare client/server applications using socket-library calls in your C program
- send/receive communication using Allen-Bradley's INTERCHANGE™ software and the INTERD daemon
- use the SNMPD daemon

For information on:	See page:
Ethernet communication	6-1
Connecting Ethernet to the network	6-2
Addresses for the Ethernet port	6-3
Modifying the Ethernet configuration files	6-4
Configuring the Ethernet port	6-12
Using the OS-9/Internet FTP utility	6-12
Using the OS-9/Internet TELNET utility	6-17
Using the Internet socket library in C programs	6-19
Using the INTERD daemon with INTERCHANGE software	6-22
Using the SNMPD daemon	6-27



Ethernet Communication

Ethernet is a local area network that provides communication between various computers (hosts) at 10 Mbps. The communication can be via thick- or thin-wire coaxial cable. The control coprocessor communicates using the OS-9/Internet software package.

OS-9/Internet provides communication between OS-9 and other Internet systems using Transmission Control Protocol/Internet Protocol (TCP/IP). OS-9 Internet C library functions provide a programming interface nearly identical to the BSD UNIX "socket" interprocess communication facilities.

OS-9/Internet provides utilities for file transfer (FTP) and terminal connection (TELNET) to remote systems on the network. OS-9/Internet also provides socket-library functions that you use to write network client/server application programs.

See the OS-9 Internet Software Reference Manual, publication 1771-6.4.11, for information on FTP and TELNET utilities and OS-9 socket library.

Important: BASIC users cannot reference socket-library functions directly. The BASIC program must fork another process written in C.

Connecting Ethernet to the Network

The Ethernet port connects to a thin-wire or a thick-wire (coaxial) network via a 15-pin transceiver or Medium Access Unit (MAU) connection. The Allen-Bradley transceivers available for this interface are:

- 5810-AXMT (thin-wire Ethernet/802.3)
- 5810-AXMH (thick-wire Ethernet/802.3)

A network with thick-wire coaxial cable can be up to 500 meters (1,525 feet) with up to 100 nodes. Thick-wire cable is referred to as “10base5” —this means that it has a 10 Mbps transmission rate, baseband, and can be 500 meters in total length.

A network with thin-wire coaxial cable can be up to 200 meters (610 feet) with up to 32 nodes. Thin-wire cable is referred to as “10base2”—this means that it has a 10 Mbps transmission rate, baseband, and can be 200 meters in total length.

The control coprocessor connects to the transceiver using either a 2.0 meter (6.5 feet) or a 15 meter (49.2 feet) transceiver cable, which is also known as an Access Unit Interface (AUI) cable. The Allen-Bradley cable numbers/kit catalog numbers are:

Catalog Number	Contents
1785-TC02/A	Thick-wire 2.0 m (6.5 ft) transceiver cable
1785-TC15/A	Thick-wire 15.0 m (49.2 ft) transceiver cable
1785-TAS/A (kit)	Thin-wire transceiver and 2.0 m (6.5 ft) cable
1785-TAM/A (kit)	Thin-wire transceiver and 15.0 m (49.2 ft) cable
1785-TBS/A (kit)	Thick-wire transceiver and 2.0 m (6.5 ft) cable
1785-TBM/A (kit)	Thick-wire transceiver and 15.0 m (49.2 ft) cable

To install the cable, attach the cable male connector to the Ethernet female connector on the control-coprocessor main module. See Appendix C for more information on cable configuration and pin assignments.

See the Installation Data, Allen-Bradley Ethernet/802.3 Transceiver, publication 5810-2.1, to connect the transceiver to your Ethernet network.

Addresses for the Ethernet Port

You must have two separate and unique addresses for the control-coprocessor Ethernet port:

- software configurable Internet Protocol (IP) address and host name that you acquire from your network manager
- hardware Ethernet address that is assigned to the control coprocessor by Allen-Bradley at the factory—see the section on configuring the Ethernet port to identify this address on page 6-11

Acquiring Your IP Address
If you are adding the control coprocessor to an existing network, contact your network administrator to get an IP address. IP addresses must be unique; each address is dependent on how your network is configured and whether the local area network has a gateway to other networks.

Important: Do not make up your own IP address. Get the number from your network administrator; your network administrator should have acquired a set of numbers from InterNIC Registration Services. If you do not have a network administrator, get your numbers from InterNIC Registration Services.

You can get IP addresses by contacting InterNIC Registration Services via electronic mail to `hostmaster@rs.internic.net`. If electronic mail is not available to you, mail a hardcopy request to:

Network Solutions
Attention: InterNIC Registration Services
505 Huntmar Park Drive
Herndon, VA 22070

You can also contact InterNIC Registration Services at 1-703-742-4777.

Acquiring a Host Name
Indicate your preference for a host name. Host names must be unique within a domain. Host names must start with a letter and be only alpha-numeric with dashes and/or underscores.

A host name consists of a word or other character string. The character string usually consists of lower-case characters. The string is typically a maximum of 8 characters. The full host name includes both the host name and the domain name. For example:

Host name: maggie
Domain: “copro.ab.com”
Full host name: “maggie.copro.ab.com”

Modifying the Ethernet Configuration Files

Make a list of all the hosts with their IP addresses that are on the same physical network as the control coprocessor. Also, make a list of all the hosts with their IP addresses that are on other networks with which you will communicate regularly through a gateway. You will use this information to update the configuration files.

The following generic network database files are included on your software installation disk in the INET directory. Using the DOS editor on your personal computer, update the files with information specific to your system.

After you have updated all the files, configure the control-coprocessor Ethernet port using the Internet Utilities of PCBridge. See page 6-12. It is not necessary to send each file to the control coprocessor separately.

Important: The BASIC user must have PCBridge to update the configuration files and configure the control-coprocessor Ethernet port.

HOSTS File

Change file C:\INET\HOSTS to include your system information. The first non-comment and non-blank line is for the loopback host and should not be changed. The lines that follow are for hosts on your network. You list the following information for each host:

- IP address
- host name
- full host name
- aliases (optional)

You should include the alias “localhost” for the control coprocessor.

You can organize the data either by frequency of use or in an order designated by your network administrator. See Figure 6.1.

Figure 6.1
List of Hosts on Network

```
$ list hosts
#

#   Sample Network
#
# The internet number can be generally anything except 0 or 255.
# If your are connecting to the DARPA internet, you already know
# what network numbers should be used.
#
127.0.0.1       loopback
130.151.128.1           me localhost
130.151.128.2           taro
130.151.128.3           grin
130.151.128.95          gespack
130.151.132.188     pc_mike
130.151.132.180     copro_12
192.52.109.1        alpha
192.52.109.2        beta
192.52.109.3        gamma
192.52.109.4        delta

$
 VT100      17:04 -CA -PR -LO +LF -LE +XO -CT  CD COM2 38400N81  PCBridge
```

HOSTS.EQUIV File

The file C:\INET\HOSTS.EQU is required for the OS-9 Internet database `INETDB` but is not used by the control coprocessor. Do not modify this file. A sample is shown in Figure 6.2.

DOS limits file names to eight characters—not including a 3-digit extension—and thus PCBridge renames the file to HOSTS.EQU. However, the OS-9 name is HOSTS.EQUIV.

Figure 6.2
List of HOSTS.EQUIV

```
$
$
$
$
$
$
$
$
$
$
$
$
$
$ dir
                          Directory of . 10:08:38
hosts           hosts.equiv     inetdb          le0_147         networks
password        protocols       services        startinet
$
$
$ list hosts.equiv
#
# used for rcp/rsh (not supported)
#
$
 VT100      10:07 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

NETWORKS File

Change file C:\INET\NETWORKS to include a line for the network to which the control coprocessor will be physically attached, giving the domain name and the subnet address of the domain. Also, include the aliases “ethernet” and “localnet.” A sample NETWORKS file is shown in Figure 6.3.

Figure 6.3
List of NETWORKS File

```
$ list networks
#
# Microware local network
#
loopback    127
os9-ether   192.52.109  ethernet localnet
sun-ether   192.9.200
test-ether  192.6.100

#
# Internet networks
#
arpanet     10      arpa
ucb-ether   46      ucbether


$




 VT100      17:02 -CA -PR -LO +LF -LE +XO -CT  CD COM2 38400N81  PCBridge
```

PROTOCOLS File

The file C:\INET\PROTOCOL is required for the Internet database. Do not change the file. An example PROTOCOLS file is shown in Figure 6.4.

DOS limits file names to eight characters, and thus PCBridge renames the file PROTOCOLS to PROTOCOL; however, the OS-9 name remains PROTOCOLS.

Figure 6.4
List of PROTOCOLS File

```
$
$
$ dir
                          Directory of . 10:10:55
hosts           hosts.equiv     inetdb          le0_147         networks
password        protocols       services        startinet
$ list protocols
#
#  "PROTOCOLS"
#
ip      0   IP      #internet protocol, pseudo protocol number
icmp    1   ICMP    # internet control message protocol
ggp     3   GGP     # gateway-gateway protocol
tcp     6   TCP     # transmission control protocol
egp     8   EGP     # exterior gateway protocol
pup     12  PUP     # PARC universal packet control
udp     17  UDP     # user datagram protocol
hmp     20  HMP     # host monitoring protocol
xns-idp 22  XNS-IDP # Xerox NS IDP
rdp     27  RDP     # "reliable datagram" protocol
nd      77  ND      # Sun ND protocol
raw     255 RAW     # is this really a protocol?
max     256 MAX     # is this really a protocol?
$
 VT100      10:10 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

SERVICES File

The file C:\INET\SERVICES contains standard Internet information followed by information that applies to your system. If you write socket-library programs to establish application-specific servers, place the “well-known” port numbers of your new services at the end of this file. We recommend that you use numbers larger than 3000 for your services. See Figure 6.5 through Figure 6.8 for an example.

The last three entries in Figure 6.8 show an example of application-specific modifications to the SERVICES file.

Figure 6.5
List of SERVICES File (Sheet 1)

```
#
#   "SERVICES"
#
# Internet port/socket assignments
#

echo        7/udp
echo        7/tcp
discard     9/udp       sink null
discard     9/tcp       sink null
systat      11/tcp
daytime     13/udp
daytime     13/tcp
netstat     15/tcp
chargen     19/tcp      ttytst source
chargen     19/udp      ttytst source
ftp-data    20/tcp
ftp         21/tcp
telnet      23/tcp
smtp        25/tcp  mail
time        37/tcp  timserver
time        37/udp  timserver
name        42/tcp  nameserver
whois       43/tcp  nicname
 VT100      10:11 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

Figure 6.6
List of SERVICES File (Sheet 2)

```
domain      53/udp
domain      53/tcp
hostnames   101/tcp     hostname    # usually to sri-nic
sunrpc      111/udp
sunrpc      111/tcp

#
# Host-specific
#

tftp        69/udp
rje         77/udp
finger      79/udp
link        87/udp      ttylink
supdup      95/udp
iso-tsap    102/tcp
x400        103/tcp             # ISO Mail
x400-snd    104/tcp
csnet-ns    105/tcp
pop         109/tcp     postoffice      # Post Office
uucp-path   117/tcp
nntp        119/tcp     usenet
ntp         123/tcp
NeWS        144/tcp     news        # Window System
 VT100      10:11 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

Figure 6.7
List of SERVICES File (Sheet 3)

```
#
# UNIX specific services
#

exec        512/tcp
login       513/tcp
shell       514/tcp     cmd     # no passwords used
printer     515/tcp     spooler # experimental
courier     530/tcp     rpc     # experimental
biff        512/udp     comsat
who         513/udp     whod
syslog      514/udp
talk        517/udp
route       520/udp     router routed
timed       525/udp     timeserver
netnews     532/tcp     readnews
uucp        540/tcp     uucpd           # uucp daemon
new-rwho    550/udp     new-who         # experimental
rmonitor    560/udp     rmonitord       # experimental
monitor     561/udp                     # experimental
ingreslock  1524/tcp

#
 VT100      10:11 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

Figure 6.8
List of SERVICES File (Sheet 4)

```
route       520/udp     router routed
timed       525/udp     timeserver
netnews     532/tcp     readnews
uucp        540/tcp     uucpd           # uucp daemon
new-rwho    550/udp     new-who         # experimental
rmonitor    560/udp     rmonitord       # experimental
monitor     561/udp                     # experimental
ingreslock  1524/tcp

#
# OS-9 specific services
#

os9srv      2600/udp    unisrv
unisrv      2600/udp    os9srv

#
# Application-specific services (for this project)
#

from_area   3001/tcp
to_area     3002/tcp
check_net   3003/tcp
$
 VT100      10:11 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

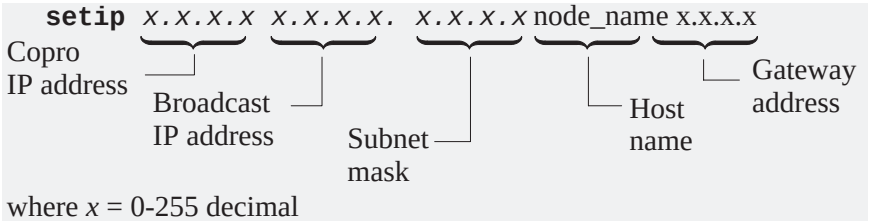
STARTINET File

The STARTINET procedure file is used to start the network.

Edit the second (`setip`) line of file C:\INET\STARTINE by inserting the following information as appropriate for your network:

- IP address
- broadcast address
- subnet mask
- host name (optional)
- target gateway address (optional)

IP Address The IP address is the only required parameter on the setip line, as shown below.



You can also invoke the setip command at the OS-9 prompt. This changes any of your Internet settings dynamically.

Broadcast Address The broadcast address specifies the range of addresses that will receive your broadcast messages. It also determines the range of addresses from which you receive broadcast messages.

The default broadcast address is your IP address with the last byte set to 255. This sets the broadcast range to only your internal subnets (no external addresses) when the subnet mask is set to 255.255.255.0.

Subnet Mask The subnet mask allows your system administrator to divide your internal network into separate subnets. The standard mask is 255.255.255.0 and allows up to 256 subnets on your internal network.

Host Name The host name allows the socket call `gethostbyname`() to return the name that you configured on the setip line as previously shown.

Target Gateway Address The target gateway address is only needed if you require immediate routing after power-up. It otherwise can take up to 30 seconds for the routing table to be initialized.

DOS limits file names to eight characters, and thus PCBridge uses the file name STARTINE. However, it is renamed STARTINET when it is sent to the control coprocessor.

The STARTINET procedure file is used to start the network. This file indicates which network daemons—e.g., FTP, TELNET, INTERD, SNMPD—are loaded at reset. See Figure 6.9 for an example STARTINET file. The example file shipped in revision 1.20 and later of the PCBridge software (1771-PCB) includes the startup of the INTERD and SNMPD daemons. You can modify this file to choose which daemons are started for your system. A line started with an asterisk (*) is a comment line.

Important: If you want to use the INTERD daemon, you must have Series A Revision E (1.30) or later of the coprocessor firmware.

Figure 6.9
List of STARTINET File

```
$ list startinet
load -d inetdb          ;* load inet database from ram-drive
load -d interd          ;* load interd from ram-drive
load -d skimp           ;* load snmpd from ram-drive
setip 130.151.128.1 130.151.228.255 255.255.255.0 taro
load -d le0_147         ;* load lance/enet descriptor from ram-drive
load -d socket          ;* load socket descriptor from ram-drive
mbinstall               ;* load and start mbuf handler
routed<>>>/nil&         ;* start up routed
* ispstart&             ;* comment since routed is running we don't need it
sleep 2                 ;* wait a couple seconds until things calm down
telnetd <>>>/nil&       ;* start daemon if we'd like to support telnet
ftpd <>>>/nil&          ;* start daemon if we'd like to support ftp
interd <>>>/nil&        ;* start daemon if we'd like to support interchange
snmpd <>>>/nil&         ;* start daemon if we'd like to support snmp
chd /dd                 ;* go to top of ram-drv directory
$
 VT100      17:04 -CA -PR -LO +LF -LE +XO -CT  CD COM2 38400N81  PCBridge
```

Password File

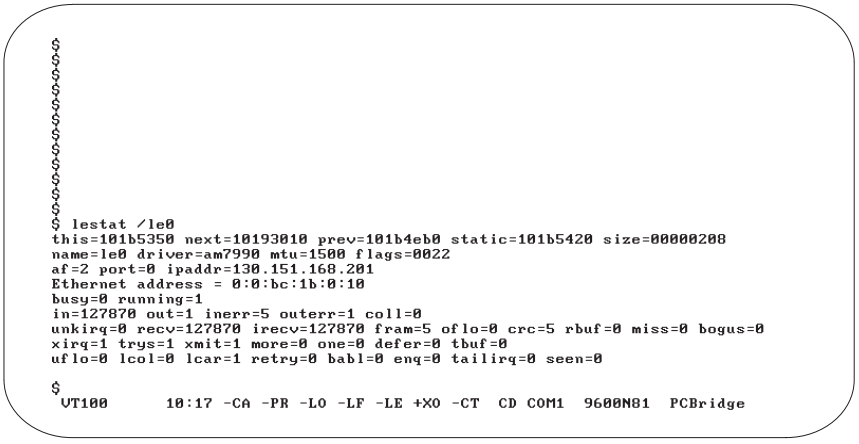
You must have a password file when using Ethernet. See the Create a User Startup File section in Chapter 3 to set up your password file.

Ethernet Hardware Address

If your network administrator requires the control-coprocessor Ethernet hardware address, then at the $ prompt, enter **`lestat /le0`**. You get the screen illustrated in Figure 6.10. The Ethernet address is shown on the fourth line in the example.

Important: The lestat utility works only after the Ethernet port is initialized by the startup procedure file.

Figure 6.10
List Ethernet Hardware Address



Configuring the Ethernet Port

To download the Ethernet configuration files to the control coprocessor and initialize the Ethernet port:

1. Select `I) Internet Utilities` on the PCBridge main menu.
2. Select `S) Setup OS-9 Internet` on the next menu.

   This causes the configuration files to be sent to the control coprocessor. These files will be stored in the SYS directory on the control-coprocessor RAM disk.
3. At the OS-9 $ prompt, type **`startinet`** and press **`[Return]`** —this invokes the STARTINET procedure file to start up the Internet software.

You can now use TELNET, FTP, and the socket library for your custom Internet applications. See the OS-9 Internet Software Reference Manual, 1771-6.4.11, for more information. Also, refer to the README file in the INET directory for more information.

Using the OS-9/Internet FTP Utility

The file transfer utility (FTP) is used to transfer files to and from remote systems. There are many FTP commands to facilitate file manipulation between systems. See the OS-9 Internet Software Reference Manual, 1771-6.4.11, for more information on the FTP utility.

FTP Send Session

The following example shows how you might conduct an FTP send session:

Important: The following send session is an example only. It represents how one network is set up and accomplishes an FTP session.

1. At the $ prompt, list the file to transfer. See Figure 6.11.

Figure 6.11
Starting FTP

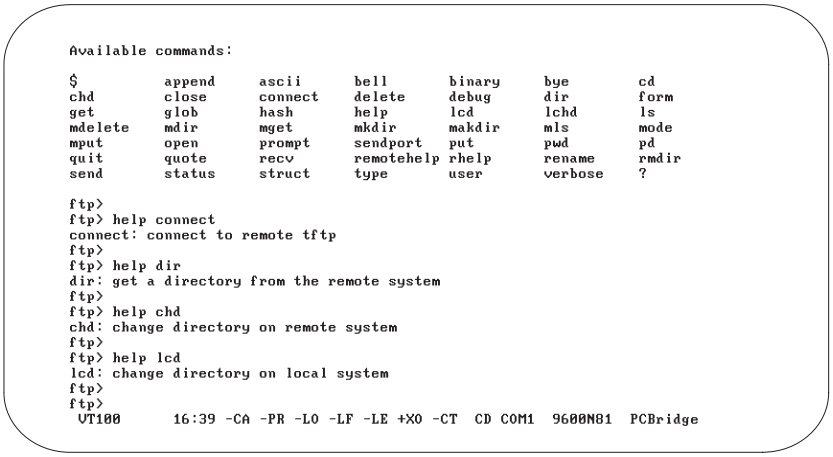
```
$
$ list hosts.equiv
#
# used for rcp/rsh (not supported)
#
$
$ ftp
Not connected.
Mode: stream       Type: ascii        Form: non-print    Structure: file
Verbose: on        Bell: off          Prompting: on      Globbing: on
Hash mark printing: off               Use of PORT commands: on
ftp>
ftp> help
Available commands:

$          append     ascii      bell       binary     bye        cd
chd        close      connect    delete     debug      dir        form
get        glob       hash       help       lcd        lchd       ls
mdelete    mdir       mget       mkdir      makdir     mls        mode
mput       open       prompt     sendport   put        pwd        pd
quit       quote      recv       remotehelp rhelp      rename     rmdir
send       status     struct     type       user       verbose    ?

ftp>
 VT100      16:38 -CA -PR -L0 -LF -LE +X0 -CT  CD COM1  9600N81  PCBridge
```

2. At the $ prompt, type **`ftp`** and press **`[Return]`**. See Figure 6.11. This utility starts the interface to the ARPANET standard File Transfer Protocol and displays the initial status.

3. At the `ftp` prompt, type **`help`** and press **`[Return]`**. See Figure 6.11. You get a list of all the available FTP commands.

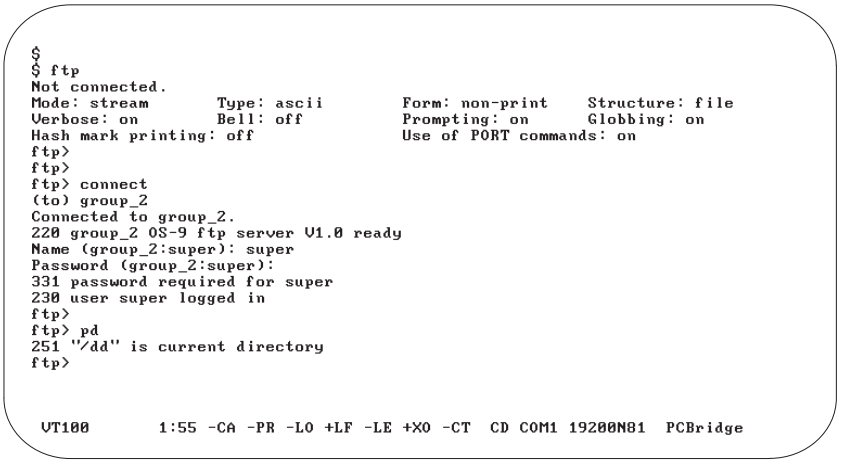
4. At the `ftp` prompt, type **`help connect`** and press **`[Return]`**. See Figure 6.12 to get information about the FTP connect command.

Figure 6.12
FTP Help



5. At the `ftp` prompt, type **`connect`** and press **`[Return]`**. See Figure 6.13.

Figure 6.13
Connect to Remote Network



6. At the `(to)` prompt , enter the name of the remote host. For this example, the remote host is `group_2`.

7. Enter user information (username and password) at the prompt. See Figure 6.13. We receive a login confirmation message.

8. At the `ftp` prompt, type **`pd`** and press **`[Return]`** for the current remote network directory name. See Figure 6.13.

9. At the `ftp` prompt, type **`chd sys`** and press **`[Return]`** to change the remote network directory. See Figure 6.14. The system response shows the new directory.

10. At the `ftp` prompt, type **`dir`** and press **`[Return]`** to get a list of files in the remote directory. See Figure 6.14.

Figure 6.14
Connect to Remote Subdirectory

```
ftp>
ftp> chd sys
200 CWD command ok
ftp>
ftp>
ftp> dir
200 PORT command ok
150 Opening data connection for dir -ea  (130.131.132.134,1028) (0 bytes).
                          Directory of . 13:51:12
 Owner    Last modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
  0.0     92/07/29 0852   ------wr       F       578 hosts
  0.0     92/07/29 0852   ------wr      10        39 hosts.equiv
  0.0     92/07/29 0852   ------wr      24      1836 inetdb
  0.0     92/07/30 1127   -----ewr       E       188 le0_147
  0.0     92/07/29 1045   ------wr      2B       134 miki
  0.0     92/07/29 0852   ------wr      17       190 networks
  0.0     92/07/30 1128   ------wr       C        80 password
  0.0     92/07/29 0852   ------wr      20       556 protocols
  0.0     92/07/29 0852   ------wr      19      1408 services
  0.0     92/07/29 0852   ------wr       9       468 startinet
226 Transfer complete
796 bytes received in 0.47 seconds (1.65 Kbytes/s)
ftp>
 VT100       1:56 -CA -PR -LO +LF -LE +XO -CT  CD COM1 19200N81  PCBridge
```

11. At the `ftp` prompt, type **`send hosts.equiv`** and press **`[Return]`**. You receive status information confirming the file transfer. See Figure 6.15.

Figure 6.15
FTP Send

```
ftp>
ftp> send hosts.equiv
200 PORT command ok
150 Opening data connection for hosts.equiv (130.131.132.134,1034).
226 Transfer complete
42 bytes sent in 0.01 seconds (4.10 Kbytes/s)
ftp>
ftp>
ftp> dir
200 PORT command ok
150 Opening data connection for dir -ea  (130.131.132.134,1035) (0 bytes).
                          Directory of . 13:53:48
 Owner    Last modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
  0.0     92/07/30 1353   ------wr      6F        39 hosts.equiv
226 Transfer complete
237 bytes received in 0.13 seconds (1.78 Kbytes/s)
ftp>
ftp>



 VT100       1:58 -CA -PR -LO +LF -LE +XO -CT  CD COM1 19200N81  PCBridge
```

12. At the `ftp` prompt, type **`dir`** and press **`[Return]`**. The remote directory now contains the transferred file. See Figure 6.15.

FTP Get Session

The following example shows how you might conduct an FTP get session.

Important: The following `ftp` get session is an example only. It represents how one network is set up and we accomplished an FTP session.

For this example session, we continue the previous FTP send session and retrieve the file we sent to a remote directory.

1. At the `ftp` prompt, type **`get`** and press **`[Return]`**. See Figure 6.16.

Figure 6.16
FTP Get

```
ftp>
ftp> get
(remote-file) hosts.equiv
(local-file) hosts.othernode
200 PORT command ok
150 Opening data connection for hosts.equiv (130.131.132.134,1045) (39 bytes).
setting hosts.othernode to 39 bytes
226 Transfer complete
42 bytes received in 0.01 seconds (4.10 Kbytes/s)
ftp>
ftp> close
200 Goodbye
ftp>
ftp> quit
$
$
$ list hosts.othernode
#
# used for rcp/rsh (not supported)
#
$


 VT100       2:03 -CA -PR -LO +LF -LE +XO -CT  CD COM1 19200N81  PCBridge
```

2. At the `(remote-file)` prompt, enter the name of the file that you want to transfer from the remote host.

3. At the `(local-file)` prompt, enter the name that you want to assign the file on the control coprocessor. You get a listing of user information confirming the transfer.

4. At the `ftp` prompt, type **`close`** and press **`[Return]`** to close the connection to the remote host. The `Goodbye` line confirms the termination of the connection.

5. At the `ftp` prompt, type **`quit`** and press **`[Return]`** to exit FTP. The OS-9 $ prompt appears.

6. At the $ prompt, type **`list hosts.othernode`** and press **`[Return]`** to view the contents of the file just received from the remote system. It is the same as the file sent.

Using the OS-9/Internet TELNET Utility

This utility provides user-interface communication to other nodes on the Internet system. The TELNET utility provides the ability to log on to remote systems using the control coprocessor and your screen as a terminal connected to the remote host. See the OS-9 Internet Software Reference Manual, publication 1771-6.4.11, for more information on the TELNET utility and commands available.

The following examples show how you might conduct a TELNET session

Important: The following TELNET session is an example only. It represents how one network is set up and how we accomplished a TELNET session.

1. At the $ prompt, type **`telnet`** and press **`[Return]`**. See Figure 6.17. This starts the interface to the `telnet` protocol.

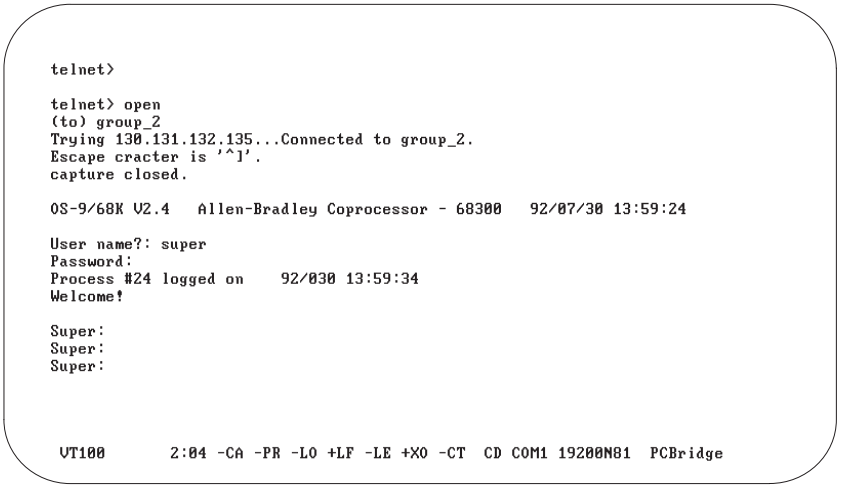
2. At the `telnet` prompt, type **`help`** and press **`[Return]`** to get a list of all the available TELNET commands. See Figure 6.17.

Figure 6.17
Starting TELNET and TELNET Help

```
$
$
$
$
$ telnet

telnet> help
close        close current connection
display      display operating parameters
mode         try to enter line-by-line or character-at-a-time mode
open         connect to a site
quit         exit telnet
send         transmit special characters ('send ?' for more)
set          set operating parameters ('set ?' for more)
status       print status information
toggle       toggle operating parameters ('toggle ?' for more)
capture      log telnet session to a file
z            fork a shell
$            fork a shell
?            print help information
help         print help information


 VT100      16:44 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

3. At the `telnet` prompt, type **`open`** and press **`[Return]`** to start a terminal session. See Figure 6.18.

Figure 6.18
TELNET Connection to Remote Network

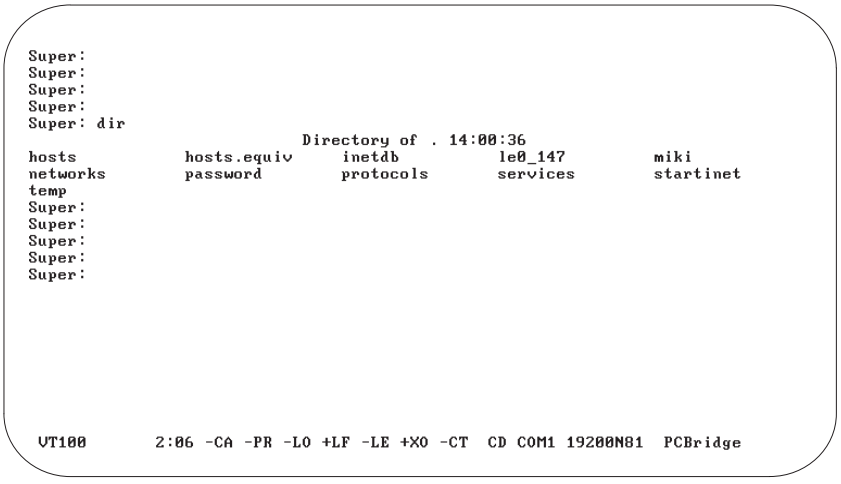


4. At the `(to)` prompt, enter the name of the the remote host to which you want to attach. For this example, the name of the remote host is `group_2`.

5. Log in and enter the account password. The response shows successful log in. See Figure 6.18.

6. At the `Super:` prompt of the remote host, type **`dir`** and press **`[Return]`** to list the directory. See Figure 6.19.

Figure 6.19
Remote Network Directory



7. At the `Super:` prompt of the remote host, type **`del host.equiv`** and press **`[Return]`**. The example deletes the file that was transferred to the remote host in the previous FTP send session. See Figure 6.20.

Figure 6.20
Telnet Delete File and Log Out

```
Super:
Super:
Super:
Super: dir
                              Directory of . 14:01:52
hosts           hosts.equiv     inetdb          le0_147         networks
password        protocols       services        startinet       temp
Super: del hosts.equiv
Super:
Super:
Super: dir
                              Directory of . 14:01:59
hosts           inetdb          le0_147         networks        password
protocols       services        startinet       temp
Super:
Super:
Super: logout
Connection closed by foreign host$
$
$


 VT100       2:07 -CA -PR -LO +LF -LE +XO -CT  CD COM1 19200N81  PCBridge
```

8. At the `Super:` prompt of the remote host, type **`dir`** and press **`[Return]`** to list the directory contents. See Figure 6.20. The file HOST.EQUIV is no longer listed in the directory.

9. At the `Super:` prompt of the remote host, type **`logout`** and press **`[Return]`** to log off.

You exited TELNET when the connection is closed and you returned to the local OS-9 $ prompt.

Using the Internet Socket Library in C Programs

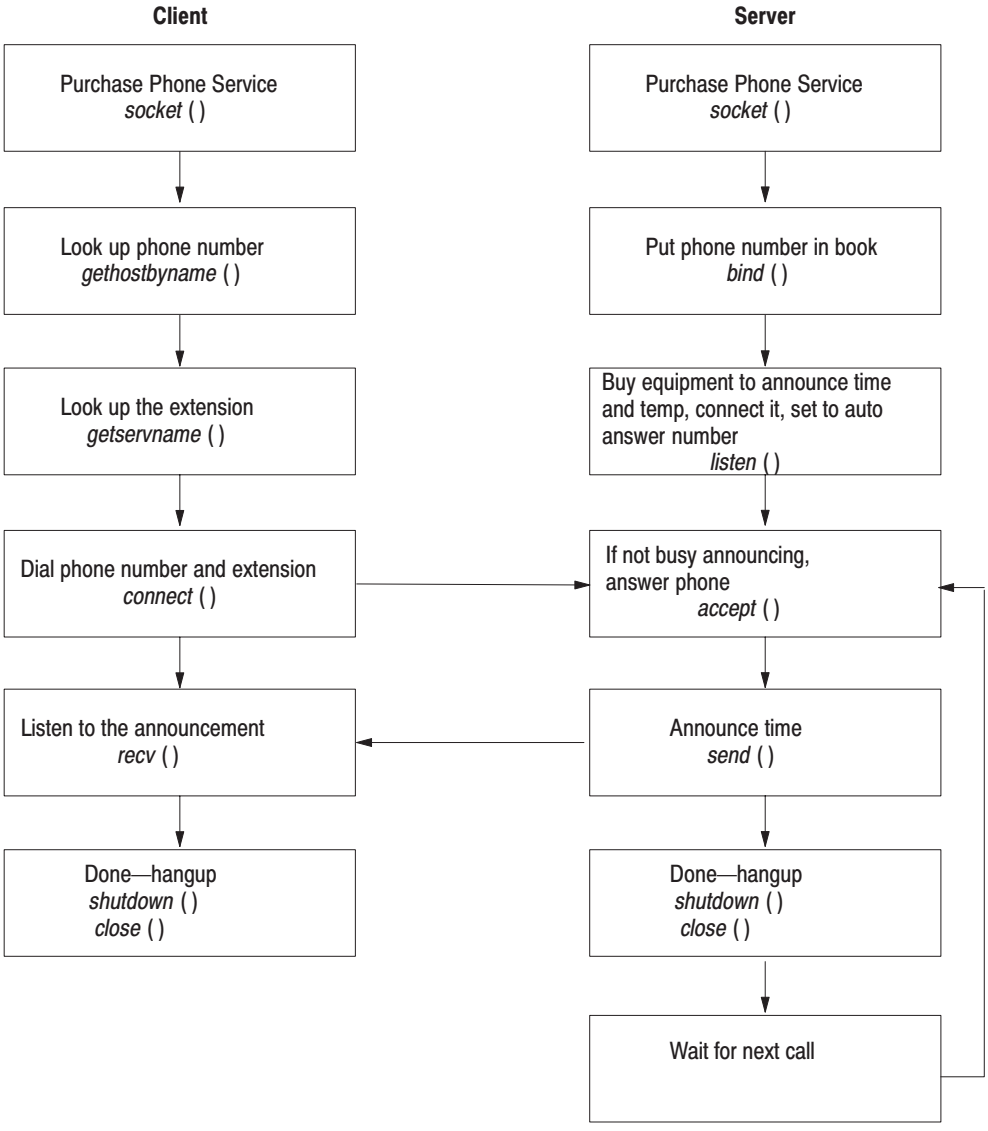
Use socket-library C calls to write client/server applications involving network data transfer. The OS-9/Internet socket library is based on the BSD-UNIX socket model for interprocess communication. This is a common method of writing client/server applications.

Analogy to Client/Server Application

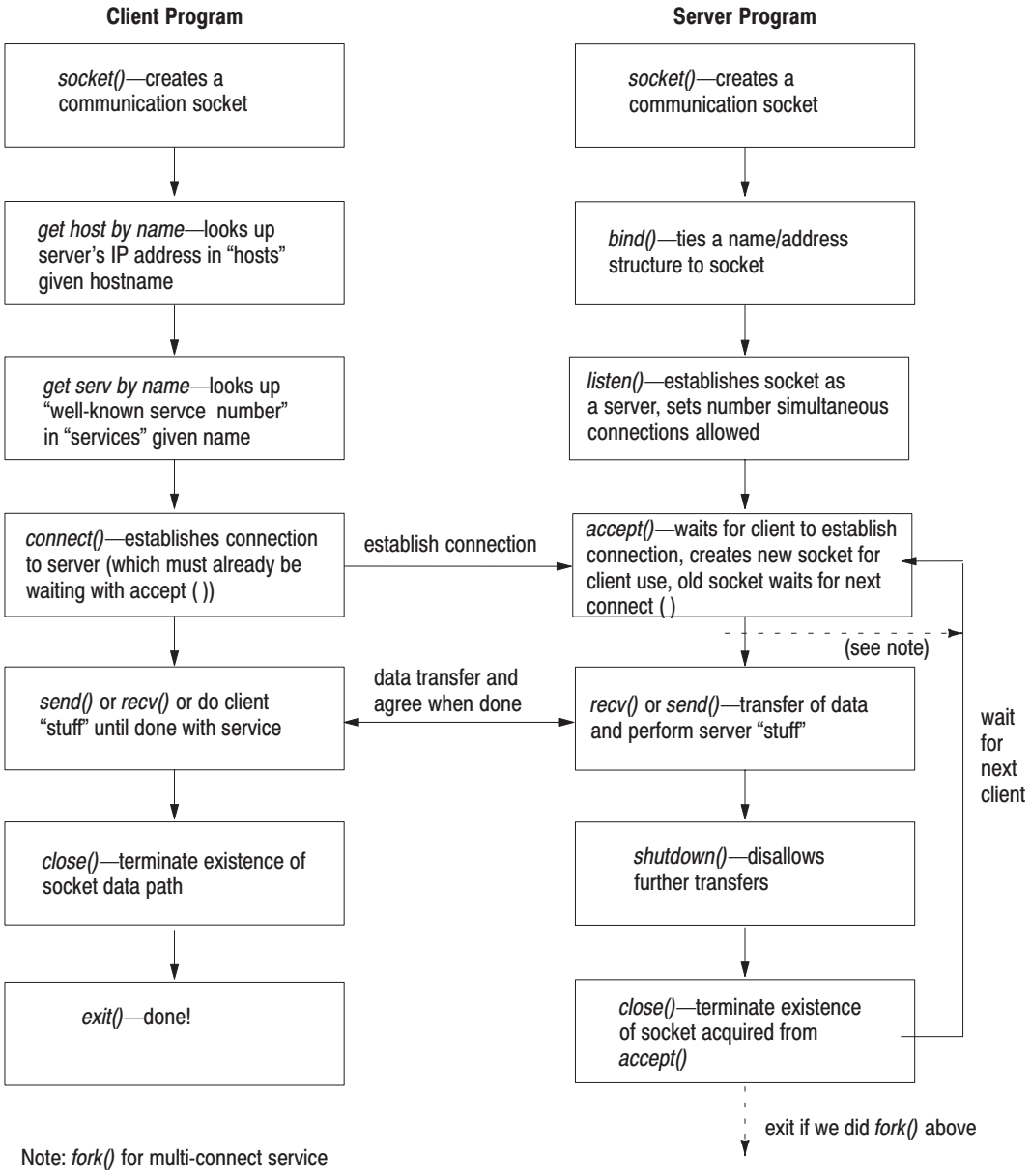
A simple analogy to the client/server application is the time-and-temperature service that is provided by the local telephone company. To acquire the time and temperature, you:

- look up `time and temperature` in the telephone directory
- dial the number
- receive an announcement of the time and local temperature
- hang up the telephone

The following flow chart shows a client/server analogy to the time-and-temperature operation.



The following flow chart shows a client/server interface.



Refer to the OS-9 Internet Software Reference Manual, publication 1771-6.4.11, Appendix B, for example programs. Each of the three different examples provide client and server programs using TCP sockets.

Using the INTERD INTERCHANGE Daemon

INTERD is a PCCC INTERCHANGE server daemon that provides communication between the control coprocessor, its attached PLC-5 processor, and a host computer running INTERCHANGE software via the Ethernet connection of the coprocessor.

INTERD is included in revision 1.20 or later of the PCBridge software (1771-PCB) and requires Series A Revision E (1.30) or later firmware in the control coprocessor.

INTERCHANGE is an Allen-Bradley application-programming interface (API) that allows easy, consistent programmatic access to control-system information.

By installing the INTERD daemon on the control coprocessor, you have the ability to do the following over the Ethernet network:

- run 6200 Series PLC-5 Programming Software on a host computer to program or monitor a PLC-5 processor connected to the coprocessor.
- run an INTERCHANGE program on a host computer to access the data table of the PLC-5 processor connected to the coprocessor.
- run an INTERCHANGE program on a host computer to access the TAG table of the coprocessor.
- access the data table of the PLC-5 processor connected to the coprocessor from a remote Ethernet PLC-5 processor by using the message instructions.

Introduction to INTERCHANGE Access to Coprocessor-Tagged Memory

You can route INTERCHANGE messages to the control coprocessor when you give the coprocessor its own station number. You do this via the CC_CFG utility. Any messages that the control coprocessor receives that are not addressed to the coprocessor are routed to the PLC-5 processor if it is connected.

INTERCHANGE accesses the TAG table of the control coprocessor using the DTL_PCCC_DIRECT function call. You read or write tagged data by issuing a PCCC typed read or write command of undefined type with an element size of 1 byte.

All external access to the control coprocessor's user memory is through the TAG table of the coprocessor. The TAG functions provide a way for you to specify access to control-coprocessor memory. The memory of the tagged area can be of any data type—e.g., char, short, float, etc.—or combination of data types. It is your responsibility to understand the layout of the tagged memory. Transmission or reception of tagged memory data is done as a "byte stream." External devices can have different memory structures—i.e., byte order, data sizes, etc. When reading tagged data from the coprocessor, the external process must accommodate the differences when interpreting the byte stream. Similarly, when writing to the tagged area, the external process must generate a byte stream to match that of the coprocessor tagged memory.

EXAMPLE Program of INTERCHANGE Access to Coprocessor-Tagged Memory

The following example illustrates how to set up the tagged memory of the coprocessor and access that memory over Ethernet using INTERCHANGE software on the host computer.

In this example, we set up a tagged area defined by the structure CAR. The TAG name "Car" points to the start of the CAR structure. The memory allocated by the OS9 compiler for the CAR structure is:

Structure	Offset	Allocation
CAR ->	00	make (bits 31-24)
	01	make (bits 23-16)
	02	make (bits 15-8)
	03	make (bits 7-0)
	04	model
	05	type
	06	color
	07	"pad" byte
	08	year (bits 31-24)
	09	year (bits 23-16)
	10	year (bits 15-8)
	11	year (bits 7-0)

Note the inclusion of a "pad" byte generated by the compiler. The pad byte is necessary to make `year` start on an even addressed boundary. This illustrates how imperative it is that you know the exact memory layout of the tagged area.

The following example of a coprocessor program creates the Car TAG and periodically increments the make, model and type elements of the structure. In this example, the coprocessor is set up to be station 22 octal (12h). To increase readability of the example, no error checking is done.

```
#include <copro.h>
typedef struct
     {
     unsigned    make;
     char        model;
     char        type;
     char        color;
     unsigned    year;
     }CAR;

  main(){
  unsigned id;                                    /* id for tag definition */
  CAR car;                                 /* car structure pointed to by tag */

  CC_INIT();                                       /* init the coprocessor */
  TAG_DEFINE (&id,&car,"Car",sizeof(car),TG_MODIFY);      /* define the tag */
  car.make = car.model = car.type = car.color = car.year = 0;  /* init data */
  while (1) {
    TAG_LOCK (id,CC_FOREVER);   /* prevent concurrent access on tagged data */
    car.make += 1;                                        /* increment make */
    car.model += 2;                                            /* and model */
    car.type += 3;                                               /* an year */
    TAG_UNLOCK (id,CC_FOREVER);                     /* allow access to tag */
    sleep (1);}                                      /* sleep for 1 second */
  }
```

The INTERCHANGE host program does the following:

1. reads and displays the entire Car TAG
2. writes a 0x99 to only the color element of the Car TAG
3. reads and displays the entire Car TAG
4. writes a 0x88 to the color element and increments the year element of the Car TAG
5. reads and displays the entire Car TAG

Note that the display routine takes the 4 bytes of the unsigned variables and places them in a temporary union variable before storing them. This—or another similar method—is necessary when the host requires that data larger than a byte be on even-address boundaries but the data for those variables in the byte stream are on odd-address boundaries.

```
#include ”dtl.h”

#define HOSTNAME ”copro2”
#define NI_ID 1

unsigned char pccc_color[] = {
  0x12,                                          /* DST  - copro station address */
  0x05,                          /* CTRL - packet type must be 5 for Interchange */
  0x00,                                 /* SRC  - Source station filled in by NI */
  0x00,                                    /* LSAP - Set to 0 for local network */
  0x0f,                                       /* CMD  - command for typed write */
  0x00,                                                /* STS  - status byte */
  0x01, 0x00,                                /* TNSW - L/H Transaction status word */
  0x67,                                             /* FNC  - typed write function */
  0x06, 0x00,                        /* OFF  - Offset L/H to requested data 6 bytes */
  0x01, 0x00,                               /* TT   - Total transaction L/H 1 item */
  0x00, ’C’,’a’,’r’,0x00,                                /* Symbolic address (TAG) */
  0x91,                                     /* Type in next byte, size of 1 byte */
  0x22,                                                    /* Undefined type */
  0x99};                                                /* Data to be transmitted */
unsigned char pccc_col_year[] = {
  0x12,                                          /* DST  - copro station address */
  0x05,                          /* CTRL - packet type must be 5 for Interchange */
  0x00,                                 /* SRC  - Source station filled in by NI */
  0x00,                                    /* LSAP - Set to 0 for local network */
  0x0f,                                       /* CMD  - command for typed write */
  0x00,                                                /* STS  - status byte */
  0x02, 0x00,                                /* TNSW - L/H Transaction status word */
  0x67,                                             /* FNC  - typed write function */
  0x06, 0x00,                        /* OFF  - Offset L/H to requested data 6 bytes */
  0x06, 0x00,                               /* TT   - Total transaction L/H 6 items */
  0x00, ’C’,’a’,’r’,0x00,                                /* Symbolic address (TAG) */
  0x99,                             /* Type in next byte, size in following byte */
  0x09,                                                      /* Type is array */
  0x03,                                                         /* of 8 bytes */
  0x91,                                     /* Type in next byte, size of 1 byte */
  0x22,                                                    /* Undefined type */
  0x88, 0xff, 0x00, 0x00, 0x00, 0x00};                  /* Data to be transmitted */
unsigned char pccc_read[] = {
  0x12,                                          /* DST  - copro station address */
  0x05,                          /* CTRL - packet type must be 5 for Interchange */
  0x00,                                 /* SRC  - Source station filled in by NI */
  0x00,                                    /* LSAP - Set to 0 for local network */
  0x0f,                                       /* CMD  - command for typed read */
  0x00,                                                /* STS  - status byte */
  0x03, 0x00,                                /* TNSW - L/H Transaction status word */
  0x68,                                             /* FNC  - typed read function */
  0x00, 0x00,                        /* OFF  - Offset L/H to requested data 0 bytes */
  0x0C, 0x00,                           /* TT   - Total transaction L/H 12 items */
  0x00, ’C’,’a’,’r’,0x00,                                /* Symbolic address (TAG) */
  0x0C, 0x00};                             /* SIZ  - Size L/H same as TT 12 items */
```

```
unsigned char pccc_rpl[275];

void main( int argc, char** argv ){
  unsigned long iostat;                          /* function completion value */
  unsigned long rpl_siz;                               /* size of pccc reply */
  DTSA_BKPLN addr;                                     /* structured address */

  DTL_INIT( 1 );                          /* Initialize the Data Table Library */

  DTL_C_CONNECT( NI_ID, HOSTNAME, 0);                              /* Connect */

  addr.atype = DTSA_TYP_BKPLN;

  addr.ni_id = NI_ID;

  rpl_siz = sizeof (pccc_rpl);                      /* size of response buffer */

  DTL_PCCC_DIRECT_W ((DTSA_TYPE *) &addr, pccc_read, sizeof (pccc_read),
    pccc_rpl, &rpl_siz, 0, 0, &iostat, 60000);              /* do typed read */

  display_tag();                                 /* show the result of the read */

  rpl_siz = sizeof (pccc_rpl);                      /* size of response buffer */

  DTL_PCCC_DIRECT_W ((DTSA_TYPE *) &addr, pccc_color, sizeof (pccc_color),
    pccc_rpl, &rpl_siz, 0, 0, &iostat, 60000);    /* typed write to color 0x99 */

  rpl_siz = sizeof (pccc_rpl);                      /* size of response buffer */

  DTL_PCCC_DIRECT_W ((DTSA_TYPE *) &addr, pccc_read, sizeof (pccc_read),
    pccc_rpl, &rpl_siz, 0, 0, &iostat, 60000);              /* do typed read */

  display_tag();             /* show the result of the read - note color = 0x99*/

  rpl_siz = sizeof (pccc_rpl);                      /* size of response buffer */

  DTL_PCCC_DIRECT_W ((DTSA_TYPE *) &addr, pccc_col_year, sizeof (pccc_col_year),
    pccc_rpl, &rpl_siz, 0, 0, &iostat, 60000);      /* write to color and year */

  rpl_siz = sizeof (pccc_rpl);                      /* size of response buffer */

  DTL_PCCC_DIRECT_W ((DTSA_TYPE *) &addr, pccc_read, sizeof (pccc_read),
    pccc_rpl, &rpl_siz, 0, 0, &iostat, 60000);              /* do typed read */

  display_tag();           /* show the result of the read - note color = 0x99 */}

display_tag (){

     /* since the pccc ”byte stream” from the coprocessor might put unsigned
        variables on uneven address boundries we move them to a temporary union
        variable before storing them */
```

```
 union
   {
    unsigned tmp;
    unsigned char c[4];
   } u;

unsigned make, year;
unsigned char model, type, color;

 u.c[0] = pccc_rpl[13];              /* get the make from the reply buffer */
 u.c[1] = pccc_rpl[14];                   /* and put it in the temp buffer */
 u.c[2] = pccc_rpl[15];
 u.c[3] = pccc_rpl[16];
 make = u.tmp;                             /* store make in make variable */
 model = pccc_rpl[17];                     /* get model from reply buffer */
 type = pccc_rpl[18];                                      /* also type */
 color =  pccc_rpl[19];                               /* as well as color */
 u.c[0] = pccc_rpl[21];        /* get the year from the reply buffer (skip */
 u.c[1] = pccc_rpl[22];        /* over pad byte at offset [21]) and put it */
 u.c[2] = pccc_rpl[23];                            /* in the temp buffer */
 u.c[3] = pccc_rpl[24];
 year = u.tmp;                             /* store year in year variable */
 u.tmp +=1;                                      /* increment year in tmp */
 pccc_col_year[25] = u.c[0];       /* move it to write data in pccc buffer */
 pccc_col_year[26] = u.c[1];
 pccc_col_year[27] = u.c[2];
 pccc_col_year[28] = u.c[3];
 printf (”make = %X model = %X type = %X color = %X year = %X\n”,
          make,model,type,color,year);          /* display the ”Car” tag */ }
```

Using the SNMPD Daemon

SNMPD is a daemon that provides Simple Network Management Protocol (SNMP) services between the control coprocessor and a host computer. This daemon supports MIB-1 variables. After installing the SNMPD daemon on the control coprocessor, you have the ability to:

- allow 6200 Series PLC-5 Programming Software to identify the coprocessor on the Ethernet network using the “WHO” function.
- monitor MIB-1 variables from a host computer running SNMP-monitoring software.

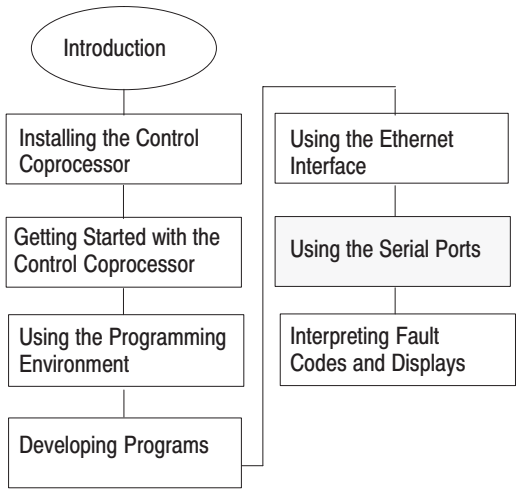
Chapter 7

Using the Serial Ports

Chapter Objectives

This chapter provides information for setting up communication with the serial ports on both the control coprocessor main module and the serial expander module.

For information on:	See page:
Setting up communication parameters	7-2
Referencing OS-9 serial port device names	7-4
Connecting to the serial port	7-4
Using a serial port for ASCII communication	7-5
Using a serial port for RS-485 communication	7-10
Using a serial port for RS-422 communication	7-17



You can use the serial ports to connect to a device that sends and receives ASCII and other serial communication.

With Series A Revision D (1.20) and later of the firmware, the serial port drivers include Data Carrier Detect functions (`_ss_dcon` and `_ss_dcoff`) and they also support RS-485 communications. The serial port on the expander module (1771-DXPS) handles reception of 7-bit even-parity communications. For more information about these functions, see the OS-9 C Language User Manual, publication 1771-6.5.104.

The serial port buffers also increased with Series A Revision D (1.20) of the firmware. The input buffers on the coprocessor and the expander increased from 80 to 128 bytes. The output buffers on the coprocessor increased from 140 to 256 bytes. The output buffers were already 256 bytes on the expander. These buffers are fixed, and they cannot be modified through user programs.

You use the PCBridge software to set up communication parameters for your personal computer. See Chapter 3.

Setting Up Communication Parameters

Prior to Series A Revision E (1.30) of the firmware, all the COMM ports on the control coprocessor and serial expander were initialized at the factory for connection to a programming terminal. These initial settings would process control-character sequences, pause characters, and program abort sequences. If you have a firmware release earlier than Series A Revision E (1.30) and you want to connect a COMM port to a device other than a terminal, you must reconfigure the port.

The 9-pin serial port COMM 0, used for configuring the coprocessor, retains the factory settings for connection to a programming terminal. In Series A Revision E (1.30) of the firmware and later, however, COMM1, COMM2, and COMM3 have all serial-port settings prepared for raw binary data transfers. For example, the XON and XOFF settings are set to 0xFF in order to turn of software and hardware handshaking; this setting allows for proper RS-485 communication.

The exact changes made to the serial port settings are listed in the following table.

Settings Prior to Series A Revision E (1.30)	Settings in Series A Revision E (1.30) and Later
noupc	noupc
bsb	nobsb
bsl	nobsl
echo	noecho
lf	nolf
null=0	null=0
nopause	nopause
pag=24	pag=0
bsp=08	bsp=00
del=18	del=00
eor=0D	eor=00
eof=1B	eof=00
reprint=04	reprint=00
dup=01	dup=00
psc=17	psc=00
abort=03	abort=00
quit=05	quit=00
bse=08	bse=00
bell=07	bell=00
type=00	type=00
baud=9600	baud=9600
xon=11	xon=FF
xoff=13	xoff=FF
tabc=09	tabc=00
tabs=4	tabs=0

ATTENTION: With `eor` and `eof` set to 0, any readln() call you make in applications never terminates. Use the raw read() call to read from the serial ports.

Use the OS-9 system management utilities `tmode` and `xmode` to set up or view the communication parameters for the control-coprocessor serial ports.

Use this mode:	To display or:
`tmode`	temporarily change the operating parameters of the current terminal session; `tmode` affects open paths and not the device descriptor; when the path is closed, the changes are lost
`xmode`	change the initialization parameters of any control coprocessor serial port; these changes will be inherited by any process that subsequently opens the path; the changes persist as long as the control coprocessor is running (even when paths to the device are repetitively opened and closed; `xmode` updates the device descriptor for the port; use the CC_VALCOMM utility (page 7-4) to make changes persist through reset/power cycles.

If you have an application that was designed prior to release 1.3 and you require the original settings of the serial port COMM1, for example, include `xmode` commands in the startup file with the following argument:

```
xmode /t1 normal
```

This returns the serial-port settings to the settings used before firmware release 1.30. See the next section for a description of CC_VALCOMM, a utility that saves the changes made after an `xmode` operation and retains the changes over power cycles.

See Figure 7.1 for the default settings for `tmode` and `xmode`.

Figure 7.1
Set Up Communication Parameters

```
$
$
$ tmode
/term
noupc bsb bsl echo lf null=0 nopause pag=24 bsp=08 del=18 eor=0D
eof=1B reprint=04 dup=01 psc=17 abort=03 quit=05 bse=08 bell=07
type=00 baud=9600 xon=11 xoff=13 tabc=09 tabs=4
$
$ xmode /t1
/t1
noupc bsb bsl echo lf null=0 nopause pag=24 bsp=08 del=18 eor=0D
eof=1B reprint=04 dup=01 psc=17 abort=03 quit=05 bse=08 bell=07
type=00 baud=9600 xon=11 xoff=13 tabc=09 tabs=4

$
$ xmode /t1 baud=19200
$
$ xmode /t1
/t1
noupc bsb bsl echo lf null=0 nopause pag=24 bsp=08 del=18 eor=0D
eof=1B reprint=04 dup=01 psc=17 abort=03 quit=05 bse=08 bell=07
type=00 baud=19200 xon=11 xoff=13 tabc=09 tabs=4

$
 VT100      16:34 -CA -PR -LO -LF -LE +XO -CT  CD COM1  9600N81  PCBridge
```

See the OS-9 Operating System User Manual, publication 1771-6.5.102, for information on the scope and lifetime of `tmode` and `xmode` changes.

Using the CC_VALCOMM Utility

Use the CC_VALCOMM utility to make `xmode` changes persist through reset and power cycles. This utility validates the device descriptor that was changed by the OS-9 `xmode` utility.

Syntax for the CC_VALCOMM utility is:

```
cc_valcomm {<opts>}
Function: Validate SCF descriptor for comm ports
Options: -comm=<num> Validate descriptor for comm port (0-3)
         -comm=*     Validate descriptor for all comm ports
```

The following example sequence sets the COMM1 port (/t1) for 19200 baud and hardware handshaking enabled .

```
$ xmode /t1 baud=19200 xon=0 xoff=0
$ cc_valcomm -comm=1
```

Hardware handshaking is enabled when the values for `xoff/xon` are both 0. The changes persist through reset and power cycles.

Referencing OS-9 Serial Port Device Names

Use the OS-9 device name as shown in the table below to reference the serial ports.

Module Port Label	OS-9 Device Name
COMM0	/term
COMM1	/t1
COMM2	/t2
COMM3	/t3

Connecting to the Serial Port

The serial ports use a data-terminal equipment (DTE) setup. Pin 2 is transmit out. Pin 3 is receive in. Pin 7 is ground. See Appendix C for information on cable connections.

Using a Serial Port for ASCII and Other Serial Communication

You can use the serial ports for ASCII and other serial communication. Examples of ASCII peripheral devices that you can use are:

- ASCII terminals
- bar-code readers
- Allen-Bradley Dataliner
- weigh scales
- printers

Accessing a Port

When you use a serial port for ASCII and other serial communication, use standard C library calls. You can use C calls such as:

- getc()
- read()
- write()

Your C program can get and change port parameters using the OS-9 library calls _gs_opt() and _ss_opt().

See the OS-9 C User Manual, publication 1771-6.5.104, for more information on C library calls.

Example Program

The following program sets up the COMM2 port (/t2) with the proper parameters for receiving data from a bar-code reader. The program:

- clears the input buffer on the COMM2 port
- loops forever, waiting for carriage-return-terminated lines of ASCII characters indicating that a bar code has been read
- places bar-code data in a buffer
- calls a subroutine to handle decision-making (based on the content of the bar code)

```
/*****************************************************************************
*   b a r c o d e . c      Bar Code Reader Interface Program
*****************************************************************************
*  LANGUAGE:  Microware C, using PCBridge release 1.4 or later
*****************************************************************************
*  TARGET SYSTEM AND VERSION:
*****************************************************************************
*     OS-9/68000 release 2.4 or later, on Allen-Bradley Control Coprocessor
*     with Ethernet and Internet Support Package (ISP)
*****************************************************************************
*  REVISION LOG:
*****************************************************************************
*    Date        By    Description
*   ----------- ----- ------------------------------------------------------
*    31-JUL-92   DER   Initial issue for User's Manual Example
*****************************************************************************
*  REFERENCES:
*****************************************************************************
*    Data Structures:     see C:\OS9C\DEFS\*.h
*                         see #include directives below for list
*    Source File:         see C:\OS9C\EXAMPLES\barcode.c
*    Linking Libraries:   see C:\OS9C\LIB\libs for list
*    Compile Commands:    see C:\PCBRIDGE\pcbcc.bat
*                                  "$ pcbcc barcode.c", then
*                                  use PCBridge to move executable
*                                  module to Copro execution directory
*    Executable Files:    see C:\OS9C\EXAMPLES\barcode
*    User's Manuals:      see Allen-Bradley Publication 1771-6.5.95,
*                             "1771 Control Coprocessor User Manual",
*                             and various Microware user's manuals
*****************************************************************************
*  USAGE:
*****************************************************************************
*    The user runs this example program from the OS-9 shell prompt ($). This program
*    displays the barcode read, then passes control to a user-developed subroutine
*    for handling the decision-making part  of the process.  A simple example is
*    provided. A typical barcode application sets up data in data modules upon a
*    successful read, rather than printing to the screen.  Modify the code here by
*    eliminating the "printf()" calls and adding calls to link to a data module or
*    write to a RAM-disk file using your barcode data.
*  EXAMPLES:
*       $ barcode <cr>
*       Waiting for a read...
*       Tag: ST0ZZ9ZZ read...
*           This is a Twinsburg product.
*       Waiting for a read...
*       Tag: SD1ZZ9ZZ read...
*           This is a Dublin product.
*       Waiting for a read... <ctrl><E>  Error 000:002
*****************************************************************************
*  INPUTS:
*****************************************************************************
*
*    port "/t2": receive barcode reader characters.  This program is set
*    up for a reader that just sends the barcode characters followed by
*    a carriage return.  IT DOES NOT HANDLE STANDARD BARCODE READER PROTOCOLS
*
*****************************************************************************
*  OUTPUTS:
*****************************************************************************
*    Screen messages as in example above.
```

```
*************************************************************************
*  FUNCTIONS CALLED BY THIS PROGRAM:
*************************************************************************
*  EXTERNAL/LIBRARY FUNCTIONS:
*
*    _gs_opt()       * get setup of serial port from OS-9
*    _gs_rdy()       * get number of characters waiting in input buffer
*    _ss_enrts()     * turn on the RTS line on the serial port
*    _ss_opt()       * change setup of serial port
*    close()         * close path to serial port
*    exit()          * abort program, exiting with status
*    memcpy()        * copy bar code data from input buffer to global data
*    open()          * open a path to the bar code reader input port
*    printf()        * print messages to terminal screen (stdout)
*    read()          * get characters from input port
*    readln()        * get <cr>-terminated line of characters from port
*
*  FUNCTIONS (defined in this file):
*    NONE
*************************************************************************
*  PROGRAMS WHICH CALL THIS PROGRAM:
*************************************************************************
*    This program is started by the user from the OS-9 shell "$" prompt
*    command line.  It loops until terminated by a user <ctrl><C> or
*    <ctrl><E>.
*************************************************************************/
#include <stdio.h>
#include <errno.h>
#include <strings.h>
#include <modes.h>
#include <sgstat.h>
#include <ctype.h>

#define  BARC_PORT        "/t2"
     /**   Makes it easy to change.  Even easier to change  ***
     ***   (but harder to program) is to read port name     ***
     ***   from command line using argc, argv.              **/
#define  BARC_BUF_SIZE    60
     /**   Not many single line readers can fit that  ***
     ***   many characters within the scan window!    **/
main ( )
{
    int             path;         /* OS-9 path number for barcode port      */
    int             sts;          /* return status of any given call...     */
    int             recd_chars;   /* number of chars gotten from barc.      */
    struct sgbuf    opts_buffer;  /* place to store OS-9 path options       */
    char            barc_buf[BARC_BUF_SIZE];  /* data buffer for barcode    */
    int             mfg_loc;      /* Manufacturing Location from barcode    */
    int             c, i, n;      /* Used to bit-bucket garbage from port   */
    while ( 1 )
    {
              /* Open a path to the serial input from the barcode reader */
              path = open ( BARC_PORT, ( S_IREAD ) );
              if ( path < 0 )
              {
              /* failure to open path to bar code reader port  */
              printf ( "BARCODE::Failed to open path to serial port!\n" );
              exit ( errno );
              }
```

```
/*  Now set up the options on the port for:
      No Echo
      No Pause
      Backspace Char = 0x7F   (DEL)
      EOF Char = 0x1A    (ctrl-z)
       7 data bits, even parity bit, 2 stop bits  */
/*  Get the options on the path, change the ones which need to be
    changed, and then send the option buffer back to the path
    descriptor  */
sts = _gs_opt ( path, &opts_buffer );
if ( sts == -1 )
      {
      printf ( "BARCODE::Can't get port options!\n" );
      exit ( errno );
      }
opts_buffer.sg_pause   =  0x00;  /*  No pause                    */
opts_buffer.sg_echo    =  0x00;  /*  No echo                     */
opts_buffer.sg_bspch   =  0x7f;  /*  DEL for bksp                */
opts_buffer.sg_eofch   =  0x1a;  /*  <ctrl><z> for EOF           */
opts_buffer.sg_parity  =  0x27;  /*  2 stop, 7 data, even parity  */
sts = _ss_opt ( path, &opts_buffer );

if ( sts == -1 )
      {
      printf ( "BARCODE::Can't set port options!\n" );
      exit ( errno );
      }
/*  Set the RTS line to the Enabled state...   */
sts = _ss_enrts ( path );
/*  Now flush the port of any garbage input from power-up.  Ditch it
    all to the bit bucket.... */
sts = _gs_rdy ( path );    /* number of chars waiting at port... */
for ( i = 0; i < sts; i++ )
    read ( path, &c, 1 );     /* throw each char. away */
/*  From here on out, we have the port; get whatever comes in from the
    bar code reader and parse it for info; then go to the appropriate
    code to distribute the data received. */
while ( 1 )
{
      /*  Read a "line" of characters from the barcode reader.
          Since the reader terminates its send with a <cr>, the
          readln() function will do the job just fine!  */
      printf ( "Waiting for a read...\n" );
recd_chars  =  readln ( path, barc_buf, BARC_BUF_SIZE );
    if ( recd_chars <= 0 )
    {
    /* end of file, or some other error...
        go back and grab port again  */
         printf ( "BARCODE::Bar Code Port EOF or Error.\n" );
         printf ( "  Attempting to reopen port...\n" );
     close ( path );
      goto reopen;
     }
    /* The last character in the buffer is a carriage return:
       overwrite it with a 0 to terminate the string...   */
    barc_buf[recd_chars-1] = 0;
         /*  Now handle the received barcode.  If some unrecoverable
             error, exit the program and tell why...          */
         printf ( "Tag: %s read...\n", barc_buf );
         sts = handle_barcode ( barc_buf );
```

```
              if ( sts != 0 )
                    {
                    printf ( “BARCODE::Error parsing barcode!\n” );
                    exit ( sts );
                    }
                    /*  Loop back to the “readln()” to get the next tag being read  */
              }
              /*  Loop back to the “open()” to try again to open path...  */
        reopen:
        ;
          }
}
int  handle_barcode ( buffer )
char    *buffer;
{
          if ( ( buffer[0] != ’S’ ) || ( !( isalpha ( buffer[1] ) ) ) )
             {
             printf ( “BARCODE::Invalid location character in barcode.\n” );
             return ( 0x0101 );
             }
    switch ( buffer[1] )
          {
            case ’D’:
                    printf ( “    This is a Dublin product.\n” );
                    break;
            case ’T’:
                    printf ( “    This is a Twinsburg product.\n” );
                    break;
            default:
                    printf ( “    I have no idea where this product came from!\n” );
                    break;
          }
          return ( 0 );
}
```

Using a Serial Port for RS-485 Communication

The control coprocessor and the expander support RS-485 communications. The modules have the necessary hardware and low-level drivers on COMM1, COMM2, and COMM3. To communicate on this network:

1. Set up the hardware.

Set the switches as shown on the side label of the coprocessor or serial expander. Connect the signal pair of wires to pins 11 and 25.

Within the control coprocessor, pin 7 is connected to the logic common of the transceiver and pin 1 is connected to chassis ground. An isolated power supply powers the module's transceiver, so there is no internal connection of the logic common to the chassis ground.

For more information, see appendix C. You can also refer to the EIA-485 standard.

The coprocessor and expander use a bidirectional transceiver. The sense of the voltages appearing across the RS-485 outputs:

- for a binary 0 (SPACE or ON) state, pin 11 is positive with respect to pin 25
- for a binary 1 (MARK or OFF) state, pin 11 is negative with respect to pin 25

You can enable and disable the transmitter (under software control) to allow it to function in typical RS-485 networks where multiple transmitters are present. You cannot disable the receiver.

The RS-485 transmitters default to transmit at power-up. If the coprocessor is connected in a multiple transmitter network, you should include a call (in the listing above) to a program that will disable the transmitter to prevent the coprocessor from disrupting the network.

2. Use the port.

Follow this order of events when writing and reading over an RS-485 network:

a. Make sure that the input buffer is empty.

b. Turn the transmitter on.

c. Write the data.

d. Use `_gs_rdy()` to verify that the data coming into the input buffer is the same as the data that was transmitted.

e. After all the data is transmitted, turn the transmitter off.

f. Clear the input buffer.

g. Wait for new data to come into the input buffer.

h. Read the new data in the input buffer.

or use the sample code on page 7-12.

There may be many transmitters and receivers connected together on an RS-485 network. If there is more than one transmitter on your network, disable the transmitter when the coprocessor is not transmitting and include collision detection. Use `_ss_enrts()` to disable the transmitter; use `_ss_dsrts()` to enable the transmitter.

You cannot disable the receiver. This means that the control coprocessor and serial expander receive all data that is transmitted and store that data in the input buffer. Make sure the input buffer does not overflow; if it does overflow, the buffer is locked until you execute an OS-9 `deiniz` and `iniz` command. Use the `_gs_rdy()` function to determine how many characters are in the input buffer.

Use the input buffer to verify that all of the data was transmitted before turning the transmitter off.

Example Code for RS-485 Communication

```
/*************************************************************************
*  i n i t _ 4 8 5 . c
*  PURPOSE: Initialize the coprocessor's serial port for generic serial
*           communication using RS-485.
*  REVISION LOG:   4/12/94         Original release of program
*                  6/30/94         Turned off psc character
*  USAGE:   This function initializes the serial port properly for doing
*           generic RS-485 communication to a coprocessor serial port.  The
*           function is passed the path of the port.  This function then
*           turns off all the default terminal settings that may adversely
*           effect communications later on.  A 0 is returned upon success and
*           a -1 if there was a problem.  Check the global variable errno
*           for the error code reported.
*  SYNOPSIS    int init_485(path)
*              int path;          - Path number from opened port
*  EXAMPLE:
*              path = open("/t1", (S_IREAD | S_IWRITE));
*              status = init_485(path);
*              if (status == -1)
*                 exit();
*************************************************************************/
/* system include files */
#include <stdio.h>
#include <sgstat.h>
int init_485(path)
int path;
{
   char buff;                    /* Character buffer */
   int status;                   /* Status variable */
   int size;                     /* Size of leftover stuff in input buffer */
   struct sgbuf opts;            /* Buffer for path descriptor information */
   /*** Get the current options ***/
   if ((status = _gs_opt(path, &opts)) == -1)
   {
      fprintf(stderr, "**** ERROR on getting port options! ***\n");
      return(-1);
   }
   /*** Set the options. This section is not needed with firmaware rel. A/E and later***/
   opts.sg_pause = 0;          /* Screen pause to off*/
   opts.sg_psch = 0;           /* No pause character */
   opts.sg_bspch = 0;          /* No backspace character */
   opts.sg_dlnch = 0;          /* No delete line character */
   opts.sg_rlnch = 0;          /* No reprint line character */
   opts.sg_dulnch = 0;         /* No duplicate last line character */
   opts.sg_tabcr = 0;          /* No tab character */
   opts.sg_echo = 0;           /* Echo off */
   opts.sg_eorch = 0;          /* Ignore end of record */
   opts.sg_eofch = 0;          /* Ignore end of file */
   opts.sg_kbach = 0;          /* Keyboard abort off - default = CTRL-E */
   opts.sg_kbich = 0;          /* Keyboard quit off - default = CTRL-C */
   opts.sg_xon = 0xff;         /* XON turned off,  0xff is special code for*/
   opts.sg_xoff = 0xff;        /* copro RS485, it will not interpret 0xff */
   opts.sg_parity = 0x00;      /* No Parity, 8 Bits, 1 Stop Bit, See below */
   opts.sg_baud = 0x0e;        /* Baud rate at 9600, See below
```

```
/*************************************************************************
*      The sg_parity is a bitfield of 8 bits.
*      Bits 0 and 1, indicate parity.         00 = no parity
*                                             01 = odd parity
*                                             11 = even parity
*      Bits 2 and 3, indicate bits/character. 00 = 8 bits/char
*                                             01 = 7 bits/char
*                                             10 = 6 bits/char
*                                             11 = 5 bits/char
*      Bits 4 and 5, indicate stop bits.      00 = 1 stop bit
*                                             01 = 1 1/2 stop bits
*                                             10 = 2 stop bits
*      Bits 6 and 7 are reserved.
*      The sg_baud is the baud rate variable (one byte field).
*      0 = 50 baud          6 = 600 baud              C = 4800 baud
*      1 = 75 baud          7 = 1200 baud             D = 7200 baud
*      2 = 110 baud         8 = 1800 baud             E = 9600 baud
*      3 = 134.5 baud       9 = 2000 baud             F = 19200 baud
*      4 = 150 baud         A = 2400 baud             10 = 38400 baud
*      5 = 300 baud         B = 3600 baud             FF = external
*************************************************************************/
   /*** Set the options ***/
   if ((status = _ss_opt(path, &opts)) == -1)
   {
      fprintf(stderr, "**** ERROR on setting port options! ***\n");
      return(-1);
   }
   /*** Make sure transmitter is off ***/
   if ((status = _ss_enrts(path)) == -1)
   {
      fprintf(stderr, "**** ERROR on disabling transmitter! ***\n");
      return(-1);
   }

   /*** Make sure buffer is empty ***/
   if ((size = _gs_rdy(path)) != -1)                  /* Is it empty? */
   {
      while (size--)
            if ((status = read(path, &buff, 1)) == -1)     /* Clear it out */
            {
               fprintf(stderr, "**** ERROR reading input buffer! ***\n");
               return(-1);
            }
   }
   return(0);  /* Everything ok */
}     /* End of function */
/*************************************************************************
*  r e a d _ 4 8 5 . c
*  PURPOSE: Read characters from a serial port configured for RS-485.
*  REVISION LOG:  4/12/94              Original release of program
*  USAGE:   This function reads characters from a port configured for RS-485.
*           Because RS-485 reads are no different than a normal read()
*           function, this read_485 function merely makes a normal read()
*           call.  This function is only included to pair with the write_485()
*           function.  The write_485() function is quite different than a
*           normal write() function.
*           Function returns the number of bytes actually read.  A -1 is
*           returned if an error occurs.  The error code is placed in the
*           variable 'errno'.
```

```
*           NOTE:  Always use a _gs_rdy() call to make sure there are
*                  is enough data to read in the input buffer before making
*                  this read call.  Otherwise the function will appear to
*                  'hang', because it is waiting for the number of characters
*                  it was told to read.
*  SYNOPSIS    int read_485(path, buffer, count)
*              int path;          - Path number from opened port
*              char *buffer;      - Pointer to buffer for read
*              int count;         - minimum size of buffer
*  EXAMPLE:    int path;
*              char in_data[10];
*              int cnt = 5;
*              path = open("/t1", (S_IREAD | S_IWRITE));
*              init_485(path);
*              status = read_485(path, in_data, cnt)
*              if (status == -1)
*                 exit();
*************************************************************************/
int read_485(path, buffer, count, timeout)
int path;
char *buffer;
int count;
int timeout;
{
   int status;
   int tmp_count=0;
   /*** Make sure there is enough data in input buffer before reading ***/
   while (((tmp_count = _gs_rdy(path)) < count) && timeout--)
         ;
   /*** Ooops, timeout.  No data.  Returning. ***/
   if (timeout == -1)
   {
      fprintf(stderr, "**** ERROR timeout in read_485() function! ***\n");
      return(-1);
   }
   /*** Do the read since there is data there ***/
   status = read(path, buffer, count);
   return(status);
}
/*************************************************************************
*  w r i t e _ 4 8 5 . c
*  PURPOSE: Write characters to a serial port configured for RS-485.
*  REVISION LOG:  4/12/94            Original release of program
*  USAGE:   This function writes characters to a port configured for RS-485.
*           Because doing serial write commands over RS-485 requires special
*           techniques to complete the write, this function was created to
*           take care of the details of transmitter control and the clearing
*           of the input buffer.
*           Function returns the number of bytes actually written.  A -1 is
*           returned if an error occurs.  The error code is placed in the
*           variable 'errno'.  The function will no return until all the
*           characters are physically transmitted out of the port.
*  SYNOPSIS    int write_485(path, buffer, count)
*              int path;          - Path number from opened port
*              char *buffer;      - Pointer to write buffer
*              int count;         - minimum size of buffer
```

```
*  EXAMPLE:    int path;
*              char out_data[3];
*              int cnt = 3;
*              path = open("/t1", (S_IREAD | S_IWRITE));
*              init_485(path);
*              status = write_485(path, out_data, cnt)
*              if (status == -1)
*                 exit();
*************************************************************************/
int write_485(path, buffer, count)
int path;
char *buffer;
int count;
{
   int tmp_count;             /* Temporary count variable */
   int status;                /* Status variable */
   int size;                  /* Size of leftover stuff in input buffer */
   int watchdog;              /* Watchdog counter variable */
   char tmp_buff[256];        /* Temporary buffer to clear input port */
   char buff;                 /* Character buffer */
   int sent;                  /* Actual number of characters sent */
   /*** Make sure buffer is empty ***/
   if ((size = _gs_rdy(path)) != -1)                 /* Is it empty? */
   {
      while (size--)
            if ((status = read(path, &buff, 1)) == -1)     /* Clear it out */
            {
               fprintf(stderr, "**** ERROR reading input buffer! ***\n");
               return(-1);
            }
   }
   /*** Enable the transmitter ***/
   if ((status = _ss_dsrts(path)) == -1)
   {
      fprintf(stderr, "**** ERROR turning on transmitter ***\n");
      return(-1);
   }
   /*** Send the data ***/
   if ((sent = write(path, buffer, count)) == -1)
   {
      fprintf(stderr, "**** ERROR writing data ***\n");
      return(-1);
   }
   /*** Watch the data going out.  When the input buffer has the same ***/
   /*** amount as sent, then the transmitter can be turned off ***/
   tmp_count = 0;
   watchdog = 50000;         /* This is worst case, it may be reduced based */
                             /* on baud rate and number of characters sent. */
   while ((tmp_count < sent) && watchdog--)
      tmp_count = _gs_rdy(path);
```

```
   /*** Report that watchdog timed out.  All characters not sent in time ***/
   if (watchdog == -1)
   {
      if (tmp_count > 0){
              if ((status = read(path, tmp_buff, tmp_count)) == -1)
              {
                  fprintf(stderr, "**** ERROR flushing input buffer ***\n");
                  return(-1);
              }
    }
      fprintf(stderr, "**** ERROR transmitting all characters - timeout ***\n");
      if (tmp_count != sent
      fprintf(stderr, "**** ERROR flushing buffer not equal to characters sent ***\n");
      return(-1);
   }
   /*** Disable the transmitter ***/
   if ((status = _ss_enrts(path)) == -1)
   {
      fprintf(stderr, "**** ERROR turning off the transmitter ***\n");
      return(-1);
   }
   /*** Flush the echo from the input buffer ***/
   if ((status = read(path, tmp_buff, tmp_count)) == -1)
   {
      fprintf(stderr, "**** ERROR flushing input buffer ***\n");
      return(-1);
   }
   return(sent);     /* Return the number of bytes sent */
}     /* End of function */
#include <modes.h>
main(argc, argv)
int argc;
char *argv[];
{
   int status;
   int x=5;
   int time=0;
   int path;
   char output[20];
   char input[25];
   int timeout;
   strcpy(output, "abcdefghijklmnopq");
   x = atoi(argv[2]);
   timeout = atoi(argv[3]);
   path = open (argv[1], (S_IREAD | S_IWRITE));
   init_485(path);
   while(x--){
      write_485(path, output, 17);
      read_485(path, input, 22, timeout);
      printf ("Received --> %s\n", input);
      tsleep(5);
   }
}
```

Using a Serial Port for RS-422 Communication

Although the control coprocessor and expander are able to communicate successfully with RS-422 devices, the RS-422 that the control coprocessor and the expander support is not true RS-422 communication. True RS-422 communication can go to 1,200 meters (3,937 ft) at the coprocessor's maximum baud rate of 19.2 kbps; the coprocessor can only go a distance of 200 ft at that rate.

True RS-422 can go long distances because it uses a balanced (differential) driver and receiver with a balanced wiring pair—i.e., a pair of wires in which neither the signal nor the signal-return line is connected to the ground—which makes it highly immune to common-mode noise, as well as because it uses a fast rise time. The coprocessor and expander use an unbalanced—single-ended—configuration where the signal-return line is connected to the ground, and the rise time is slower. A balanced transmitter/receiver can be connected to and communicate with an unbalanced transmitter/receiver as long as the maximum cable length is reduced to that of the unbalanced specification—in this case, 200 ft.

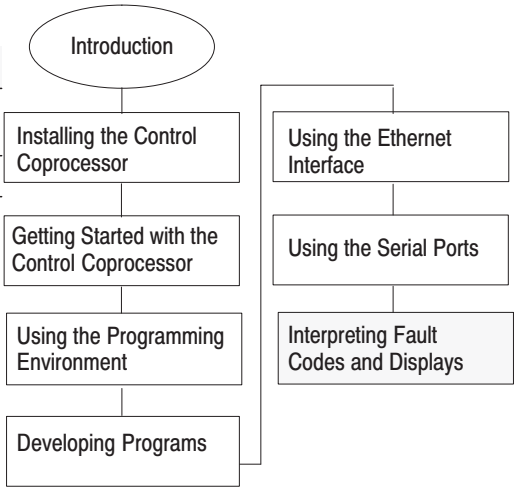
Chapter 8

Interpreting Fault Codes and Displays

Chapter Objectives

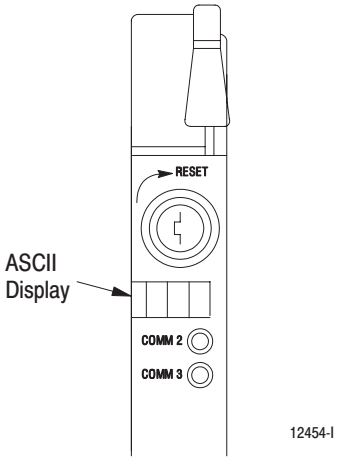
This chapter provides information on the status of the main module and the serial expander module. The LEDs on the module front panel indicate status. On the serial expander module, the user can identify faults on the ASCII display.

For information on:	See page:
Serial expander module fault display	8-1
Status for LEDs	8-2



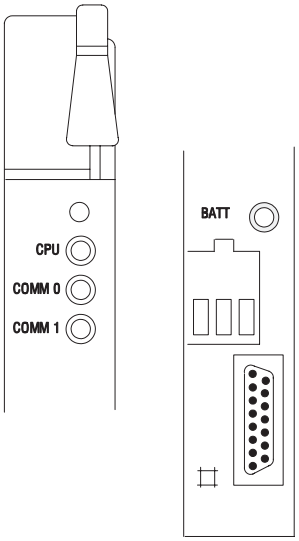
Serial Expander Module ASCII Display

The ASCII display on the serial expander module shows user-selected characters. You can configure the display using the control-coprocessor CC_DISPLAY functions that are explained in Chapter 5.



Status for LEDs

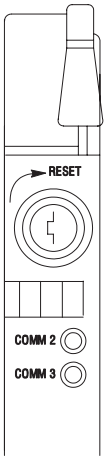
The following tables provide information for the LEDs on the control-coprocessor main module and the serial expander module.



LEDs on the Main Module

LED	Color	Indication
CPU	green	valid processor cycles are occurring
	red	a fault condition
COMM0 and COMM1 (except RS-485)①	green	receiving data
	red	transmitting data
	off	idle
COMM1 (RS-485 only)①	solid green	idle
	flickering green	receiving data
	off	transmitting data
BATT	off	good battery
	red	replace battery (or no battery installed)

① When nothing is attached to the communication port, the indicator is always green. When a device is connected to the communication port, the indicators are lit/off as indicated above.



LEDs on the Serial Expander Module

LED	Color	Indication
COMM2 and COMM3 (except RS-485)①	green	receiving data
	red	transmitting data
	off	idle
COMM2 and COMM3 (RS-485 only)①	solid green	idle
	flickering green	receiving data
	off	transmitting data

① When nothing is attached to the communication port, the indicator is always green. When a device is connected to the communication port, the indicators are lit/off as indicated above.

Appendix A

Control-Coprocessor Specifications

Product Specifications

Table A.1 lists general specifications for the control coprocessor.

Table A.1
Control-Coprocessor Specifications

Backplane Current	Main module	• 2.50 Amps at +5 Vdc (1771-DMC module with no Ethernet) • 4.00 Amps at +5 Vdc (1771-DMC1 or -DMC4 module with Ethernet and transceiver)①
	Serial expander module	1.5 Amps at +5 Vdc
Fault Relay	Serial expander module	500 mA at 30 Vac/dc (resistive)
Environmental Conditions	Operating temperature	0-60° C (32-140° F)
	Storage temperature	-40-85° C (-40-185° F)
	Relative humidity	5-95% (without condensation)
Time-of-Day Clock and Calender	Maximum variations at 60° C	±5 minutes per month
	Typical variations at 20° C	±20 seconds per month
Communication Ports②	COMM0	RS-232C; 9-pin
	COMM1, COMM2, and COMM3	RS-232C, -423, -485, and -422A compatible; 25-pin
	Ethernet port	TCP/IP protocol using FTP, TELNET, and socket library routines; INTERCHANGE server, SNMP compatible (MIB I); 15-pin standard transceiver
Communication Rates	COMM0, COMM1, COMM2, and COMM3 ports	110, 150, 300, 600, 1200, 2400, 4800, and 9600 bps, 19.2 Kbps, and 38.4 Kbps
	Ethernet	10 Mbps
Location	1771-I/O chassis	• direct-connect to a PLC-5 programmable controller • same chassis as a programmable controller, but standalone • remotely located from a programmable controller and standalone
Keying	Main module (on the upper C connector)	• between 24 and 26 • between 30 and 32
	Serial expander module (one on the upper C and two on the lower D connectors)	• between 16 and 18 (upper C connector) • between 2 and 4 (lower D connector) • between 16 and 18 (lower D connector)
Agency Certification	(Only when product is marked)	• CSA certified • CSA Class I, Division 2, Groups A, B, C, D • UL listed
Battery Life	Main module	1 year

① This is an approximate value. See Chapter 2, Installing the Control Coprocessor, for instructions on calculating backplane current requirements.

② With the 1771-DMC module (256 Kbyte), DF1 is not available on the communication ports; if you add an optional 1- or 4-Mbyte SIMM, however, the communication ports will initialize with DF1 capability.

Product Compatibility

Table A.2 lists products compatible with the control coprocessor.

Table A.2
Other Allen-Bradley Products Compatible with the Control Coprocessor

Programmable Controllers	I/O Chassis	Adapter Modules	Terminals or Personal Computers
Direct-Connect Mode • PLC-5/11 processor • PLC-5/20 processor • PLC-5/20E processor • PLC-5/30 processor • PLC-5/40 (series B, revision B or later) processor • PLC-5/40E processor • PLC-5/40L processor • PLC-5/60 (series B, revision B or later) processor • PLC-5/60L processor • PLC-5/80 processor • PLC-5/80E processor **Standalone Mode** in a programmable-controller chassis • any PLC-5 processor • mini PLC-2 processor **Standalone Mode** in a remote chassis • any PLC-5, PLC-5/250 processor • PLC-3 processor • PLC-2 processor (remote I/O only)	Any Universal 1771 I/O chassis	Any 1771-ASB adapter module in a remote chassis	**Terminals:** • VT220 (DEC) • other ASCII terminal **Personal Computers:** • IBM® PC/AT • T47 • T50 • T53 • T60

Control-Coprocessor Memory

Table A.3 shows RAM configuration. You can configure free user portion of RAM for your requirements. You can also change the default size of the TAG table. See Chapter 3 for more information.

Table A.3
RAM Configuration

Total RAM	Default RAM-Disk Size	RAM Required by the System	Default TAG Table①	Free User RAM②
256 Kbytes (1771-DMC)③	64 Kbytes	120 Kbytes	0	72 Kbytes
1 Mbyte (1771-DMC1)		215 Kbytes	80 Kbytes	665 Kbytes
4 Mbytes (1771-DMC4)		215 Kbytes	80 Kbytes	3737 Kbytes

① To change the default size of the TAG table, see the section in Chapter 3 on configuring the control coprocessor (CC_CFG utility).

② To configure the control coprocessor RAM free user memory, see Chapter 3 on configuring the control-coprocessor system memory (MEM_CFG utility).

③ Source debugging for C programs does not work with this memory configuration. Debugging requires a minimum of 512 Kbytes.

Table A.4 lists the optional RAM single inline memory modules (SIMMs) that you can add to your control coprocessor.

Table A.4
Additional RAM Memory (optional)

Memory Size	Catalog Number
256 Kbyte	1771-DRS
1 Mbyte	1771-DRS1
4 Mbytes	1771-DRS4

① These optional RAM SIMMS are **not** the same as those used in generic personal computers, which are dynamic RAM. The 1771-DRS RAM SIMMS are special static RAM chips.

Important: If you have an optional memory module (1771-DRS1 or -DRS4) in the second SIMM socket of the coprocessor and the memory in that socket is configured for battery backup, the memory in the second socket might become corrupted after power is cycled. This problem does not occur in the standard factory configuration for the 1771-DMC, -DMC1, and -DMC4. It only affects memory in the second SIMM socket on older coprocessor modules with these part numbers:

1771-DMC	96102271	96102274	96102277	96845871	96845872	96845873
1771-DMC1	96102272	96102275	96102278	96845971	96845972	96845973
1771-DMC4	96102273	96102276	96102279	96846071	96846072	96846073

If you have one of the above part numbers and you want to install and configure battery backup in the second SIMM socket, return your module to Allen-Bradley for an update.

CSA Certification

CSA certifies products for general use as well as for use in hazardous locations. Actual CSA certification is indicated by the product label. See the CSA Hazardous Location Approval Supplemental Product Information, publication ICCG-4.1, for more information.

UL Certification

Underwriters Laboratories Inc. (UL) performs safety investigations of electrical and electronic equipment and products as well as other equipment and products. After product samples have been safety tested and are found to comply with applicable safety requirements, UL authorizes a manufacturer to apply the appropriate UL Mark on products that continue to comply with the requirements. In the case of

Allen-Bradley's control coprocessor, it is the presence of the UL Listing Mark on the individual product that indicates UL certification.

Appendix **B**

Application Program Interface Library of Functions

Appendix Objectives

This appendix provides information on the Application Program Interface (API) library of functions. For each function available, you are given the following:

- C syntax
- parameters
- returns
- description
- C example
- BASIC example
- references

What Is the Application Program Interface

API is a library of functions and executable commands. The following are the functions and commands available in the API library.

API Function	Definition
BPI	Control-coprocessor commands that access the data-table memory of a programmable controller through the backplane interface (BPI)
CC	Control-coprocessor utility commands that handle functions such as trap initialization, error handling, ASCII displays, etc.
DTL	Data-table library (DTL) commands that access the data-table memory of a programmable controller that is directly connected (direct-connect mode) to the control coprocessor
MSG	Control-coprocessor message (MSG) commands that handle unsolicited Message Instructions from a programmable controller ladder-logic program (direct-connect mode)
TAG	Control-coprocessor commands (TAG) that provide access to the control-coprocessor memory for external devices that are connected via the serial interface(s); ControlView® is an example of such a device that would require access to control-coprocessor memory; TAG also provides access to control-coprocessor memory between OS-9 program modules

See Chapter 5, “Planning Programs for the Control Coprocessor”, for information on how to use these API functions in your C, assembler, and BASIC programs.

Using Pointers

The C syntax section in this appendix provides, in C definition format, the arguments for each function. In keeping with how functions are defined in C, the syntax uses an asterisk (*) in the type declaration for an argument to indicate that the function expects a pointer to the given type, for example:

```
unsigned DTL_DEF_AVAIL (num_avail)
unsigned *num_avail;
/*{a function definition would be here!} */
```

In actual practice, **the pointer must point to some existing memory**. If the declaration above were used in an actual program, the pointer, `num_avail`, would not point to anything.

To emphasize this and provide a guide for use, the C example section for each function will declare a variable of the required type (which allocates memory for the variable) and will then use the “address of” (&) operator to pass the address.

```
main ()
{
     unsigned num_avail;     /* allocated memory for variable! */
     CC_INIT ();
     .
     .
     .
     DTL_DEF_AVAIL (&num_avail);
     if (num_avail >0)        /* OK */
     .
     .
     .
}
```

When a function requires a pointer to a type, ensure that you pass the address of an area of memory that you have created.

```
char *pointer_only;                /* NO */
char allocated_array [100];        /* YES! */
char *pointer_only = “string constant has an address \n”
/* YES */
```

When a function expects a pointer to any type or to a type that depends on other arguments, the C syntax definition uses the pointer-to-void syntax.

```
void *ptr_to_some_unknown_type;
```

Your program should pass a pointer to a type that matches the data being processed.

BPI_DISCRETE

Gets the updated output-image word and optionally sets the input-image word.

Important: Only a single task should use the BPI functions. A second calling process is put to sleep if the BPI is already in use. The second task could time out unexpectedly.

C Syntax

```
#include <copro.h>

unsigned short BPI_DISCRETE  (mode, input_img)
     int mode;
     unsigned short input_img;
```

Parameters

`mode`

Specifies whether or not the function sends a new value to the programmable controller input image table using NO_MODIFY or MODIFY (defined in COPRO.H).

Important: If the mode parameter is invalid (i.e. does not equal MODIFY or NO_MODIFY), then the function will force the mode to NO_MODIFY. It is difficult to return an error in this case because the expected return is the output-image value from the programmable-controller data table.

`input_img`

The value sent to the programmable-controller input-image table at the address corresponding to the control coprocessor's physical address (according to rack and slot).

Returns

Variable	Meaning
out_img	The programmable controller's output-image word for the control coprocessor

Description

Use the BPI_DISCRETE function to return the programmable controller's updated output-image word for the control-coprocessor backplane I/O slot. The function can also update the control-coprocessor input-image word in the programmable-controller input image using the `input_img` parameter.

The only bits available for use by the application program are the upper 8 bits (10-17). The lower 8 bits (0-7) are reserved for block transfer, even if there are no block transfers programmed to the control coprocessor.

C Example

```
unsigned short output_img;
unsigned short input_img;
int mode = MODIFY
.
.
.
input_img = 0xF800;                              /* bit pattern to PLC-5
                                                 controller input word */
output_img = BPI_DISCRETE (MODE, input_img);      /* send to inpt image,
                                                  get from output image*/
if (output_img & 0x0100)
   do_bit_0_true_function ( );.
.
.
.
```

BASIC Example

The BASIC function code is 32.

Important: For BASIC, the data type for the `input_img` and `output_img` parameters is INTEGER.

```
DIM inputimg,outputimg            : INTEGER
.
.
.
inputimg=256
.
.
RUN AB_BAS(32,outputimg,1,inputimg)
.
.
.
```

References

BPI_READ(); BPI_WRITE();

BPI_READ

Responds to a synchronous block-transfer write from a programmable controller.

Important: Only a single task should use the BPI functions. A second calling process is put to sleep if the BPI is already in use. The second task could time out unexpectedly.

C Syntax

```
#include <copro.h>

unsigned BPI_READ (size,dst,timeout,trg_mask)
     unsigned char size;
     short *dst;
     unsigned int timeout;
     unsigned short trg_mask;
```

Parameters

`size`

Determines how many words the programmable controller will send.

`dst`

Provides the address of the buffer where the data is stored that the programmable controller will send.

`timeout`

The function timeout in seconds. The application program stops until the function completes or times out. A value of 0 causes the function to use the minimum value of 1 second. The maximum value is TOMAX (0x 3fff).

`trg_mask`

Use this word mask to inform a programmable controller to initiate a block-transfer write to the control coprocessor. The ladder-logic program in the programmable controller monitors this mask as a condition to trigger the block transfer. The bit mask is the actual input-image word for the rack and slot location of the control coprocessor. If the parameter is null, then it won't attempt to send the bit mask to the programmable controller before the BPI_READ.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
118	CC_E_TIME	I/O operation did not complete in time
160	CC_E_INV_TO	Invalid timeout value
164	CC_E_INV_BPI_MASK	Invalid value for BPI trigger mask
190	CC_E_SIZE	Invalid size for operation

Description

The BPI_READ routine allows the programmable controller to perform a block-transfer write over the I/O backplane to the control coprocessor.

It may accomplish a block-transfer write with the control coprocessor by allowing both a timeout and a trigger mask to be specified. The function will first do a single transfer with the programmable controller using the caller's trigger mask. The function will then return to the caller when either the block transfer or the timeout occurs. The function will return a success or fail status. In the case of a fail status, the caller can check the returned status value to find out why the function failed (i.e., transfer_size > 64).

The programmable-controller ladder program can monitor the input image for the control coprocessor to receive the trigger mask. If one of the user-specified bits goes true, then a block-transfer write to the control coprocessor can be initiated.

C Example

```
unsigned char   size=6;        /* size of block transfer */
short           inbuff[32];    /* user location to copy data to */
unsigned int    timeout=2;     /* user timeout in two seconds */
unsigned short  trg_mask=0x400; /* trigger mask, bit 12 (octal) is set */
int             status;        /* status value of BPI_READ */
.
.
.
status = BPI_READ  (size, inbuff, timeout, trg_mask);
.
.
.
```

BASIC Example

The BASIC function code is 34.

Important: For BASIC, the data type for the `inbuff` and `trgmask` parameters is INTEGER. There is no byte-type constant; therefore, byte-type variables must be used to pass the byte-type information.

```
DIM status,timeout,inbuff(32),trgmask          : INTEGER
DIM size                                       : BYTE
.
.
.
timeout=4
trgmask=4
size=6
.
.
RUN AB_BAS(34,status,size,ADDR(inbuff(1)),timeout,trgmask)
.
.
.
```

References

BPI_WRITE();

BPI_WRITE

Responds to a synchronous block-transfer read from a programmable controller.

Important: Only a single task should use the BPI functions. A second calling process is put to sleep if the BPI is already in use. The second task could time out unexpectedly.

C Syntax

```
#include <copro.h>

unsigned BPI_WRITE (size,src,timeout,trg_mask)
     unsigned char size;
     short *src;
     unsigned int timeout;
     unsigned short trg_mask;
```

Parameters

`size`

Determines how many words the programmable controller will receive.

`src`

Provides the address of the buffer containing the data that the programmable controller will receive.

`timeout`

Function timeout in seconds. The application program sleeps until the function completes or times out. A value of 0 causes the function to use the minimum value of 1 second. The maximum value is TOMAX (0x 3fff).

`trg_mask`

Use this word mask to inform a programmable controller to initiate a block-transfer read to the control coprocessor. The ladder-logic program in the programmable controller monitors this mask to trigger the block transfer. The bit mask is the actual input-image word for the rack and slot location of the control coprocessor. If the parameter is null, then it won't attempt to send the bit mask to the programmable controller before the BPI_WRITE.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
118	CC_E_TIME	I/O operation did not complete in time
160	CC_E_INV_TO	Invalid timeout value
164	CC_E_INV_BPI_MASK	Invalid value for BPI trigger mask
190	CC_E_SIZE	Invalid size for operation

Description

Use the BPI_WRITE routine to allow the programmable controller to perform a block-transfer read over the I/O backplane to the control coprocessor.

It may accomplish a block-transfer read with the control coprocessor by allowing both a timeout and a trigger mask to be specified. The function will first do a single transfer with the programmable controller using the caller's trigger mask. The programmable-controller ladder program can monitor the input image for the control coprocessor to receive the trigger mask. If one of the user-specified bits goes true, then a block-transfer read to the control coprocessor is initiated. The function will then return to the caller when either the block transfer or the timeout has occurred. This function will return a success or fail status. In the case of a fail status, the caller can check the returned status value to find out why the function failed—i.e., transfer_size > 64.

C Example

```
unsigned char   size=6;           /* size of block transfer */
short           outbuff[32];      /* location of copy data from */
unsigned int    timeout=2;        /* user timeout in two seconds */
unsigned short trg_mask = 0x400;  /*trigger mask, bit 12 (octal) is set */
int status;                       /*status value of BPI_WRITE*/
.
.
.
status = BPI_WRITE  (size, outbuff, timeout, trg_mask);
.
.
.
```

BASIC Example

The BASIC function code is 33.

Important: For BASIC, the data type for the `outbuff` and `trgmask` parameters is INTEGER. There is no byte-type constant; therefore, byte-type variables must be used to pass the byte-type information.

```
DIM status,timeout,outbuff(32),trgmask              : INTEGER
DIM size                                            : BYTE
.
.
.
timeout=4
trgmask=2
size=6
.
.
RUN AB_BAS(33,status,size,ADDR(outbuff(1)),timeout,trgmask)
.
.
.
```

References

BPI_READ();

CC_DISPLAY_DEC

Displays an integer value in decimal on the ASCII display of the serial expander module.

C Syntax

```
#include <copro.h>

unsigned CC_DISPLAY_DEC (val)
     int val;
```

Parameters

`val`

Contains the integer value to be displayed in decimal form on the ASCII display. The display ranges from -999 to 9999.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
141	CC_E_CNVT	Data-conversion error
159	CC_E_NOEXPANDER	Expander not present

Description

Use the CC_DISPLAY_DEC function to display an integer value in the ASCII display. The value must be in the range -999 through 9999.

Important: ASCII display remains unchanged until another display function call is performed successfully.

C Example

```
unsigned status;.
int value = 1234;
.
.
.
status = CC_DISPLAY_DEC (value);
.
.
.
```

BASIC Example

The BASIC function code is 106.

```
DIM status            : INTEGER
DIM data              : INTEGER
.
.
.
rem * CC_DISPLAY_DEC - Display data to the expander as 4
rem *                  decimal characters
RUN AB_BAS (106,status,data)
.
.
.
```

References

CC_ERROR(); CC_ERRSTR();

CC_DISPLAY_EHEX

Displays an unsigned-integer value in hexadecimal on the ASCII display of the serial expander module.

C Syntax

```
#include <copro.h>

unsigned CC_DISPLAY_EHEX (val)
    unsigned val;
```

Parameters

`val`

Contains the unsigned-integer value to be displayed in hexadecimal on the ASCII display. The display ranges from 0 to FFFF.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
141	CC_E_CNVT	Data-conversion error
159	CC_E_NOEXPANDER	Expander not present

Description

Use the CC_DISPLAY_EHEX function to display an unsigned-integer value in 4-digit hexadecimal on the ASCII display. The value must be in the range of 0 through FFFF.

Important: ASCII display remains unchanged until another display function call is performed successfully.

Be consistent when using hexadecimal or decimal radix on the ASCII display for ease of interpretation—i.e., some hexadecimal values can appear to be decimal values.

C Example

```
unsigned status;.
.
.
status = CC_DISPLAY_EHEX (0x301F);
.
.
.
```

BASIC Example

The BASIC function code is 105.

```
 DIM status            : INTEGER
 DIM data              : INTEGER
.
.
.
rem * CC_DISPLAY_EHEX - Display data to the expander as 4
rem *                   hexadecimal characters
RUN AB_BAS (105,status,data)
.
.
.
```

References

CC_ERROR(); CC_ERRSTR();

CC_DISPLAY_HEX

Displays an unsigned-integer value in hexadecimal on the ASCII display.

C Syntax

```
#include <copro.h>

unsigned CC_DISPLAY_HEX (val)
     unsigned val;
```

Parameters

`val`

The unsigned-integer value to be displayed in hexadecimal on the ASCII display. The display ranges from 0H to FFFH.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
141	CC_E_CNVT	Data-conversion error
159	CC_E_NOEXPANDER	Expander not present

Description

Use the CC_DISPLAY_HEX function to display an unsigned-integer value in 3-digit hexadecimal on the ASCII display. The value must be in the range of 0H through FFFH. The 3 digits are displayed with a trailing “H.”

Important: ASCII display remains unchanged until another display function call is performed successfully.

C Example

```
unsigned status;.
.
.
status = CC_DISPLAY_HEX (0x301);
.
.
.
```

BASIC Example

The BASIC function code is 104.

```
DIM status            : INTEGER
.
.
.
rem * CC_DISPLAY_HEX - Display data to the expander as 3
rem *                  hexadecimal characters followed by H
RUN AB_BAS (104,status,$345)
.
.
.
```

References

CC_ERROR(); CC_ERRSTR();

CC_DISPLAY_STR

Copies four characters to the ASCII display.

C Syntax

```
#include <copro.h>

unsigned CC_DISPLAY_STR (str_ptr)
     char *str_ptr;
```

Parameters

`str_ptr`

Specifies a pointer to the buffer that contains the characters to display.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
159	CC_E_NOEXPANDER	Expander not present

Description

Use the CC_DISPLAY_STR function to display a 4 character string on the optional ASCII display.

Important: ASCII display remains unchanged until another display function call is performed successfully.

CC_DISPLAY_STR looks at the string as a four-character buffer. Therefore, it is not necessary to include a null character as a terminator. Likewise, any null character occurring within the four-character buffer will be displayed.

C Example

```
unsigned status;
char buff [4];
.
.
.
buff[0] = 0x02;
buff[1] = '5';
buff[2] = '5';
buff[3] = 0x02;
while  (1)
{
          status = CC_DISPLAY_STR (buff);
          sleep  (2);
          status = CC_DISPLAY_STR ("Fred");
          sleep  (2);
}
```

BASIC Example

The BASIC function code is 102.

```
DIM status            : INTEGER
.
.
.
rem * CC_DISPLAY_STR - Display the string -AB- on expander module
RUN AB_BAS (102,status,"-AB-")
.
.
.
```

References

CC_ERROR(); CC_ERRSTR();

CC_ERROR

Gets a pointer to a NULL-terminated “canned” error message.

C Syntax

```
#include <copro.h>

char *CC_ERROR (error)
     unsigned error;
```

Parameters

`error`

Specifies the error message to print. The number is typically supplied by the status returned from an API function call or the I/O status returned in iostat.

Returns

Variable	Value
str_ptr	str_ptr is a pointer to the “canned” error message; see Table B.A for a list of all the error messages

Description

Use the CC_ERROR function to get a pointer to the “canned” error message corresponding to an error number. This error number is typically the value of the iostat variable or the return status of an API function. We recommend that you use this function in a C routine, although you can use it in a BASIC procedure. See CC_ERRSTR for a BASIC procedure.

C Example

```
unsigned status;
unsigned machine1;
unsigned iostat;
unsigned short parts1;
char *err_stg;
.
.
.
status = DTL_READ_W (machinel, &parts1, &iostat);
if (status != DTL_SUCCESS || iostat !=DTL_SUCCESS)
{
         err_str = CC_ERROR (status)
         printf (“Error during read : %s - status = %d\n”,err_str,status);
         err_str = CC_ERROR (iostat)
         printf (“Error during read : %s - iostat = %d\n”, err_str,iostat);
}
.
.
.
```

BASIC Example

The BASIC function code is 100.

```
DIM ptr              : INTEGER
DIM iostat           : INTEGER
.
.
.
rem * CC_ERROR - Get the pointer to the string for the iostat value
RUN AB_BAS (100,ptr,iostat)
print using “h8”,ptr
.
.
.
```

References

CC_ERRSTR();

CC_ERRSTR

Copies the “canned” null-terminated error message into the user’s local buffer.

C Syntax

```
#include <copro.h>

void CC_ERRSTR (error,err_ptr)
     unsigned error;
     unsigned char *err_ptr;
```

Parameters

`error`

Specifies which error message to copy. The number is typically supplied by the status returned from an API function call or the I/O status returned in iostat.

`err_ptr`

This parameter specifies an 80-character buffer to which the error string will be copied.

Returns

None.

Description

Use the CC_ERRSTR function to copy the “canned” error message corresponding to an error number into the user’s local buffer. This error number is typically the value of the iostat variable or the return status of an API function. We recommend that you use this function in a BASIC procedure, although you can use it in a C routine. See CC_ERROR for a C routine.

C Example

```
unsigned status, mach1, iostat;
unsigned short part1;
char err_txt[80];
.
.
.
status = DTL_READ_W (mach1, &part1, &iostat);
if (status != DTL_SUCCESS || iostat != DTL_SUCCESS)
{
      CC_ERRSTR (status, err_text);
      printf (“Read error %d: %s\n”, status, err_text);
      CC_ERRSTR (iostat, err_text);
      printf (“I/O status %d: %s\n”, iostat, err_text);
}
else
      printf (“Read: SUCCESS!\n”);
```

BASIC Example

The BASIC function code is 101.

```
DIM buffer           : STRING[81]
DIM status           : INTEGER
DIM iostat           : INTEGER
.
.
.
rem * CC_ERRSTR - Get the string for the iostat value - display on terminal
RUN AB_BAS (101,status,iostat,ADDR(buffer))
print buffer
.
.
.
```

References

CC_ERROR();

CC_EXPANDED_STATUS

Gets current expanded status information of the coprocessor.

C Syntax

```
#include <copro.h>

unsigned CC_EXPANDED_STATUS (exp_stat);
     unsigned *exp_stat;
```

Parameters

`exp_stat`

A pointer to a buffer of 5 unsigned integers that receive the expanded status information.

Buffer:	With this status information:
[0]	Total Memory
[1]	TAG Table Size
[2]	NV Disk Size
[3]	NV Module Memory
[4]	NV User Memory

Returns

Value	Meaning
xx	Bitmap of the current coprocessor status

See CC_STATUS for definition of the bit map.

Description

Use CC_EXPANDED_STATUS to get the current expanded status information of the coprocessor.

C Example

```
unsigned coprostat;
unsigned exp_stat[5];
.
.
.
coprostat = CC_EXPANDED_STATUS(exp_stat);
if (!(coprostat & 0x0001)) BAT_LOW_ALARM ();
printf (“NV Module Memory Size is %x\n”,ext_stat[3]);
.
.
.
```

BASIC Example

The BASIC function code is 112.

```
DIM coprostat                   :INTEGER
DIM extstat(5)                  :INTEGER
.
.
.
rem * CC_EXPANDED_STATUS - Get current expanded coprocessor
rem *                      status information
RUN AB_BAS (112,coprostat,ADDR(extstat))
print using “S20<,H8”, “NV Module Memory   = ”,extstat(3)
.
.
.
```

References

CC_STATUS(); utility cc_status

CC_GET_DISPLAY_STR

Copies the characters of the current ASCII display to the user's buffer.

C Syntax

```
#include <copro.h>

unsigned CC_GET_DISPLAY_STR (str_ptr)
     char *str_ptr;
```

Parameters

`str_ptr`

Specifies a pointer to users buffer to receive the display characters. This function always copies four characters. No null is appended.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
159	CC_E_NOEXPANDER	Expander not present

Description

Use the CC_GET_DISPLAY_STR function to get the current display values.

C Example

```
unsigned status;
char buff [4];
.
.
.
status = CC_GET_DISPLAY_STR (buff);
.
.
.
```

BASIC Example

The BASIC function code is 103.

```
DIM status            : INTEGER
DIM dspbuff(4)        : BYTE
.
.
.
rem * CC_GET_DISPLAY_STR - Get display data from the expander display
RUN AB_BAS (103,status,ADDR(dspbuff(1)))
.
.
.
```

References

CC_ERROR(); CC_ERRSTR();

CC_INIT

Initializes internal data structures and installs trap handler.

Important: The CC_INIT function must be called before you can use any API function.

C Syntax

```
unsigned CC_INIT()
```

Parameters

None.

Returns

None.

Description

Use the CC_INIT function to initialize internal control-coprocessor memory structures and install the trap handler used for the user's API functions.

C Example

```
main ()
{
CC_INIT ();
}
.
/* other API functions */
.
.
.
```

BASIC Example

```
procedure COPRO
rem * CC_INIT - This call must be made before any other API functions are called
RUN AB_BAS (0)
.
.
.
```

CC_PLC_BTR

Requests the PLC-5 programmable controller to perform a block-transfer read from an intelligent I/O module.

Important: You can use this function **only** if the coprocessor is connected directly to the PLC-5 programmable controller.

C Syntax

```
# include <copro.h>

unsigned CC_PLC_BTR (r,g,m,size,retry,data_ptr,iostat)
     unsigned char r;
     unsigned char g;
     unsigned char m;
     unsigned char size;
     unsigned char retry;
     unsigned short *data_ptr;
     unsigned *iostat;
```

Parameters

`r`

The assigned rack number in which the target I/O module resides.

`g`

The I/O group number that specifies the target I/O module.

`m`

The module slot number within the I/O group.

`size`

The number of words to be read from the I/O module.

`retry`

The retry value for doing the block transfer. If the value is, 0 the processor will retry the transfer one time before returning. If the value is 1, the processor will repeatedly attempt the transfer from an unresponsive module for four seconds.

`data_ptr`

The address of a data buffer that will store the block transfer read data.

`iostat`

This parameter returns a final completion status. Possible completion status values are shown in the following table.

Value	Meaning
0	CC_SUCCESS = operation completed successfully
127	CC_E_NOATMPT = I/O operation not attempted; see status variable for reason
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
124	CC_E_FAIL	Expander not present
157	CC_E_NOTCONNECT	PLC is not connected or offline
165	CC_E_BAD_RACK	Rack value out of range
166	CC_E_BAD_GROUP	Group value out of range
167	CC_E_BAD_MODULE	Module slot value out of range
168	CC_E_BAD_RETRY	Retry value out of range
190	CC_E_SIZE	Invalid size for operation

Description

Use the CC_PLC_BTR function to get block-transfer information from an analog I/O module. This function may take a long period of real time to complete.

C Example

```
unsigned char rack = 0;
unsigned char group = 5;
unsigned char module = 0;
unsigned char size = 1;
unsigned iostat;
unsigned short buff;
.
.
.
status = CC_PLC_BTR (rack,group,module,size,1,&buff,&iostat);
if (!(status))printf (“value from module is %x\n”,buff[0]);
.
.
.
```

BASIC Example

Important: For BASIC, the data type for the buff parameter is INTEGER. There is no byte-type constant; therefore, byte-type variables must be used to pass the byte-type information.

The BASIC function code is 114.

```
DIM apistat                    : INTEGER
DIM iostat                     : INTEGER
DIM buff                       : INTEGER
DIM r                          : BYTE
DIM g                          : BYTE
DIM m                          : BYTE
DIM s                          : BYTE
DIM rt                         : BYTE
.
.
.
r:=0
g:=5
m:=0
s:=1
rt:=1
rem * CC_PLC_BTR - Get block transfer information from I/O
RUN AB_BAS (114,apistat,r,g,m,s,rt,ADDR(buff),ADDR(iostat))
print using “S20<,H8”, “Value from module = ”, buff
.
.
.
```

References

CC_PLC_BTW();

CC_PLC_BTW

Requests the PLC-5 programmable controller to perform a block-transfer write to an intelligent I/O module.

Important: You can use this function **only** if the coprocessor is connected directly to the PLC-5 programmable controller.

C Syntax

```
# include <copro.h>

unsigned CC_PLC_BTW (r,g,m,size,retry,data_ptr,iostat)
     unsigned char r;
     unsigned char g;
     unsigned char m;
     unsigned char size;
     unsigned char retry;
     unsigned short *data_ptr;
     unsigned *iostat;
```

Parameters

`r`

The assigned rack number in which the target I/O module resides.

`g`

The I/O group number that specifies the target I/O module.

`m`

The module slot number within the I/O group.

`size`

The number of words to be written to the I/O module.

`retry`

The retry value for doing the block transfer. If the value is 0, the processor will retry the transfer one time before returning. If the value is 1, the processor will repeatedly attempt the transfer to an unresponsive module for four seconds.

`data_ptr`

The address of a data buffer that contains the block-transfer write data.

`iostat`

This parameter returns a final completion status. Possible completion status values are shown in the following table.

Value	Meaning
0	CC_SUCCESS = operation completed successfully
127	CC_E_NOATMPT = I/O operation not attempted; see status variable for reason
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
124	CC_E_FAIL	Expander not present
141	CC_E_CNVT	Data-conversion error (BASIC only)
157	CC_E_NOTCONNECT	PLC is not connected or offline
165	CC_E_BAD_RACK	Rack value out of range
166	CC_E_BAD_GROUP	Group value out of range
167	CC_E_BAD_MODULE	Module slot value out of range
168	CC_E_BAD_RETRY	Retry value out of range
190	CC_E_SIZE	Invalid size for operation

Description

Use the CC_PLC_BTW function to put block-transfer information to an analog I/O module. This function may take a long period of real time to complete.

C Example

```
unsigned char rack = 0;
unsigned char group = 5;
unsigned char module = 0;
unsigned char size = 1;
unsigned iostat;
unsigned short buff;
.
.
.
buff = 0x23;
status = CC_PLC_BTW (rack,group,module,size,1,&buff,&iostat);
if (!(status))printf (“data sent to module\n”);
.
.
.
```

BASIC Example

Important: For BASIC, the data type for the buff parameter is INTEGER. There is no byte-type constant; therefore, byte-type variables must be used to pass the byte-type information.

The BASIC function code is 113.

```
DIM apistat                    : INTEGER
DIM iostat                     : INTEGER
DIM buff                       : INTEGER
DIM r                          : BYTE
DIM g                          : BYTE
DIM m                          : BYTE
DIM s                          : BYTE
DIM rt                         : BYTE
.
.
.
r:=0
g:=5
m:=0
s:=1
rt:=1
buff :=$23
rem * CC_PLC_BTW - Put block transfer information to I/O
RUN AB_BAS (113,apistat,r,g,m,s,rt,ADDR(buff),ADDR(iostat))
print “data sent to module”
.
.
.
```

References

CC_PLC_BTR();

CC_PLC_STATUS

Returns current status of the processor status flags and major fault words. This function can be used with a direct-connect mode control coprocessor only.

C Syntax

```
#include <copro.h>

unsigned CC_PLC_STATUS (plc_sts)
     unsigned *plc_sts;
```

Parameters

`plc_sts`

A bit map of the current PLC-5 programmable controller status. The bit map is defined as:

Bit No.	Definition	Bit No.	Definition
0	RAM bad	16	bad user program memory
1	run mode	17	illegal operand address
2	test mode	18	programming error
3	program mode	19	function chart error
4	burning EEPROM	20	duplicate labels found
5	download mode	21	power loss fault
6	edits enabled	22	chan 3 fault
7	remote modes	23	user jsr to fault routine
8	forces enabled	24	watchdog fault
9	forces present	25	system illegally configured
10	successful EEPROM burn	26	hardware fault
11	online editing	27	MCP file does not exist /MCP file is not ladder or SFC
12	debug mode	28	PII file is not ladder /PII file does not exist
13	user program checksum done	29	STI program is not ladder /STI program does not exist
14	last scan of ladder/SFC step	30	Fault program is not ladder /fault program does not exist
15	first scan of ladder/SFC step	31	Faulted program does not exist /fault program is not ladder

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use CC_PLC_STATUS to get the current PLC-5 programmable-controller status.

C Example

```
unsigned status;
unsigned plc_sts;
.
.
.
status = CC_PLC_STATUS (&plc_sts);
.
.
.
```

BASIC Example

The BASIC function code is 108.

```
 DIM status            : INTEGER
 DIM plc_stat          : INTEGER
.
.
.
rem * CC_PLC_STATUS - Get current PLC status information
RUN AB_BAS (108,status,ADDR(plc_stat))
print using “S16<,H8”, “PLC status     = ”, plc_stat
.
.
.
```

References

None.

CC_PLC_SYNC

Synchronize with PLC-5 program scan. This function can be used with a direct-connect mode control coprocessor only.

C Syntax

```
#include <copro.h>

unsigned CC_PLC_SYNC ( )
```

Parameters

None.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the CC_PLC_SYNC function to synchronize to the PLC-5 programmable-controller ladder scan. This routine will put the calling task to sleep until the PLC-5 programmable controller signals the start of a new ladder scan. Due to the multitasking of OS-9, it should be noted that this function is most effective when only one task is synchronized to the PLC-5 programmable-controller scan and that task is a higher priority than the other tasks.

C Example

```
unsigned status;
.
.
.
status = CC_PLC_SYNC();
.
.
.
```

BASIC Example

The BASIC function code is 107.

```
DIM status            : INTEGER
.
.
.
rem * CC_PLC_SYNC - synchronize to the PLC ladder scan
RUN AB_BAS (107,status)
.
.
.
```

References

None.

CC_STATUS

Returns current status information of the coprocessor.

C Syntax

```
#include <copro.h>

unsigned CC_STATUS ();
```

Parameters

none

Returns

Value	Meaning
xx	Bitmap of the current coprocessor status

The bit map is defined as:

Bit No.	Definition
0	Battery Status (0=low, 1=ok)
1	PLC-5 On-Line Status (0=off-line, 1=on-line)
2	Expander Presence Status (0=not present, 1=present)
3	PLC-5 Reset Enable Status (0=disabled, 1=enabled)
4	Not used
5	Flash Test Status (0=failed, 1=ok)
6-7	Not used
8-11	Encoded Memory Size (in megabytes) 1 = 0.25 2 = .0.50 3 = 1.00 4 = 1.25 5 = 2.00 6 = 4.00 7 = 4.25 8 = 5.00 9 = 8.00
12-15	Not used
16-23	RAM Disk Size (in number of 64Kb blocks)
24-31	Station Address

Description

Use CC_STATUS to get the current status information of the coprocessor.

C Example

```
unsigned coprostat;
.
.
.
coprostat = CC_STATUS();
if (!(coprostat & 0x0001)) BAT_LOW_ALARM ();
.
.
.
```

BASIC Example

The BASIC function code is 111.

```
DIM coprostat                    :INTEGER
.
.
.
rem * CC_STATUS - Get current coprocessor status information
RUN AB_BAS (111,coprostat)
print using “S20<,H8”,“Coprocessor status = ”,coprostat
.
.
.
```

References

CC_EXPANDED_STATUS(); utility cc_status

DTL_C_DEFINE

Adds a definition to the DTL data-definition table.

C Syntax

```
#include <copro.h>

unsigned DTL_C_DEFINE (name_id,data_definition)
     unsigned *name_id;
     char *data_def;
```

Parameters

`name_id`

Use to return a handle assigned by the library to the data.

`data_definition`

Use to specify the data you wish to access. The data_definition character string is a null-terminated string composed of arguments separated by commas.

`“data_address,[elements],[CC data_type],[access type]”`

`data_address`

Specifies the starting address of the data item.

The first three data files in the PLC-5 programmable controller are fixed. When addressing them, **do not** reference a file number. Use `I:03` for rack 0 group 3, for example, not `I1:03` for file number 1.

`[elements]`

Optional; specifies the number of consecutive data elements, starting at data_address, to be included in the data item. The number of elements multiplied by the number of bytes per element must be <= 2000 bytes. **Default is 1 element**.

You can specify elements to the bit level—for example, `B3:/4` would point only to bit 4.

`[CC data_type]`

Optional; specifies data type of calling programs copy of the data.

CC Data Type:	Is:
RAW	no conversion
BYTE	char (signed)
UBYTE	unsigned char
WORD①	short (signed)
UWORD	unsigned short

CC Data Type:	Is:
LONG	int (signed)
ULONG	unsigned int
FLOAT	float
DOUBLE	double

① Default is WORD

`[access type]`

Optional; legal access rights are:
* READ = read only
* MODIFY = read or write

Default is modify.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
03	DTL_E_DEFBAD2	Invalid number of elements to DEFINE
04	DTL_E_DEFBAD3	Invalid data type
05	DTL_E_DEFBAD4	Invalid access rights
09	DTL_E_DEFBADN	Invalid number of DEFINE parameters
11	DTL_E_FULL	Data DEFINE table is full
16	DTL_E_INVTYPE	Data is invalid type for operation
19	DTL_E_NOINIT	DEFINE table not initialized
31	DTL_E_TOOBIG	Data item greater than maximum allowed
38	DTL_E_DFBADADR	Bad DEFINE address
40	DTL_E_INPTOOLONG	DEFINE input string too long

Description

Use the DTL_C_DEFINE function to add a data definition to the table of data definitions for the calling task. The DTL_C_DEFINE routine returns a handle with which the calling task can refer to the data item in subsequent DTL calls. You must use the DTL_C_DEFINE function to create an entry for each contiguous range of data-table locations you need.

C Example

```
unsigned fred;      /*handle used in later DTL_READ_W or
                                        DTL_WRITE_W calls*/
unsigned status;
status = DTL_C_DEFINE (&fred,“N10:2,10,WORD,READ”);
```

BASIC Example

The BASIC function code is 2.

```
procedure COPRO
DIM status   : INTEGER
DIM fred     : INTEGER
.
.
.
rem * DTL_C_DEFINE - Define a data element
RUN AB_BAS (2,status,ADDR(fred),“N10:2,10,LONG,MODIFY”)
.
.
.
```

References

DTL_READ_W(); DTL_WRITE_W(); DTL_INIT();
DTL_RMW_W(); DTL_DEF_AVAIL();

DTL_CLOCK

Sets the control-coprocessor date and time to the same date and time found in the PLC-5 programmable controller.

Syntax

```
#include <copro.h>

unsigned DTL_CLOCK ()
```

Description

DTL_CLOCK synchronizes the control coprocessor time to within one second of the clock for the PLC-5 programmable controller. This is a one-time-only synchronization. The user can maintain synchronization by executing DTL_CLOCK at regular intervals.

Since this routine performs I/O to the PLC-5 programmable controller, the calling process must call DTL_INIT prior to calling DTL_CLOCK.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
18	DTL_E_TIME	I/O operation did not complete in time
19	DTL_E_NO_INIT	DEFINE table not initialized
42	DTL_E_GETIME	PLC-5 time invalid

C Example

```
unsigned status;
status = DTL_INIT (1);
status = DTL_CLOCK ();
```

BASIC Example

The BASIC function code is 18.

```
procedure COPRO
DIM status       : INTEGER
.
.
.
rem * DTL_CLOCK - synchronize our clock with the PLC-5
RUN AB_BAS (18,status)
.
.
.
```

References

DTL_INIT();

DTL_DEF_AVAIL

Returns the number of data definitions that can be added to the DTL data-definition table.

C Syntax

```
#include <copro.h>

unsigned DTL_DEF_AVAIL (num_avail)
     unsigned *num_avail;
```

Parameters

`num_avail`

Contains the number of data definitions remaining in the data-definition table.

Description

Use the DTL_DEF_AVAIL function to determine the number of data definitions available in the calling task's table of data definitions. The function calculates the difference between the number of entries defined by DTL_INIT and the number of successful data definitions made using DTL_C_DEFINE and returns the results in the num_avail parameter.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NO_INIT	DEFINE table not initialized

C Example

```
unsigned status;
unsigned num_avail;
.
.
.
status = DTL_DEF_AVAIL (&num_avail);
printf ("%d definitions available\n",num_avail);
```

BASIC Example

The BASIC function code is 4.

```
procedure COPRO
DIM status    : INTEGER
DIM num_avail : INTEGER
.
.
.
rem * DTL_DEF_AVAIL - How many definitions available
RUN AB_BAS (4,status,ADDR(num_avail))
.
.
.
```

References

DTL_C_DEFINE(); DTL_C_UNDEF();

DTL_GET_FLT

Gets a floating-point value from a byte array.

C Syntax

```
#include <copro.h>

unsigned DTL_GET_FLT (in_buf,out_val)
     unsigned char *in_buf;
     float *out_val;
```

Parameters

`in_buf`

Use to specify an array of four bytes containing an IEEE floating-point value read from the data table as raw data.

`out_val`

Contains the floating point value. It is assumed that the bytes are read into the a data area using the RAW data_type.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful

Description

Use the DTL_GET_FLT to converts raw 32-bit IEEE float data, in 4-byte array, to host-type float.

C Example

```
float read_val;
unsigned char untyped_data[60];
unsigned machine, iostat;
.
.
.
DTL_C_DEFINE (&machine, “F8:10,15,RAW,READ”);
if (DTL_READ_W (machine, untyped_data, &iostat) == 0)
{
DTL_GET_FLT (&untyped_data [3], &real_val);
}
```

BASIC Example

Important: For BASIC, the data type for float_val is REAL.

The BASIC function code is 9.

```
procedure COPRO
DIM status       : INTEGER
DIM floatbuff(4) : BYTE
DIM float_val    : REAL
.
.
.
rem * DTL_GET_FLT
RUN AB_BAS (9,status,ADDR(floatbuff(1)),ADDR(float_val))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_3BCD();
DTL_GET_4BCD(); DTL_GET_WORD();
DTL_PUT_FLT(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_GET_WORD

Gets a word from a byte array.

C Syntax

```
#include <copro.h>

short DTL_GET_WORD (in_buf)
     unsigned char *in_buf;
```

Parameters

`in_buf`

Use to specify an array of two bytes containing programmable-controller data.

Returns

Variable	Value
word_val (short)	word_val is the value generated by combining the two bytes into one word.; the bytes are assumed to have been read into the data area using the RAW qualifier

Description

Use the DTL_GET_WORD function to extract two bytes from a byte array in programmable-controller format (raw) and returns a word (short) value in control-coprocessor format.

C Example

```
short word_val;
unsigned char untyped_data[50];
unsigned machine;
unsigned iostat;
.
.
.
DTL_C_DEFINE (&machine, “N7:0,25,RAW,READ”);
if (DTL_READ_W (machine, untyped_data, &iostat)==DTL_SUCCESS)
{
          word_val = DTL_GET_WORD (&untyped_data[11]);
}
```

BASIC Example

Important: For BASIC, the data type for the word_val parameter is INTEGER.

The BASIC function code is 8.

```
procedure COPRO
DIM word_val     : INTEGER
DIM getbuff(2)  : BYTE
.
.
.
rem * DTL_GET_WORD
RUN AB_BAS (8,word_val,ADDR(getbuff(1)))
.
.
.
```

References

DTL_GET_FLT(); DTL_GET_3BCD();
DTL_GET_4BCD(); DTL_GET_WORD();
DTL_PUT_FLT(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_GET_3BCD

Gets a 3-digit BCD value from a byte array.

Important: This function only examines the low-order 12 bits of the buffer containing the BCD value. Data in the high-order 4 bits are ignored when converting to binary format.

C Syntax

```
#include <copro.h>

unsigned DTL_GET_3BCD (in_buf,out_val)
     unsigned char *in_buf
     unsigned *out_val
```

Parameters

`in_buf`

Use to specify an array of two bytes that contains the 3-digit BCD value. It is assumed the data were read from a data item with a control-coprocessor data type that is raw.

`out_val`

Contains the binary value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
41	DTL_E_CNVT	Data conversion error

Description

Use the DTL_GET_3BCD to convert a programmable-controller 3-digit BCD value, stored in a 2-byte array, to a control-coprocessor unsigned value.

C Example

```
unsigned status;
unsigned char thumbwheel_data [2];
unsigned thumbwheel_binary;
.
.
.
status = DTL_GET_3BCD (thumbwheel_data,&thumbwheel_binary);
```

BASIC Example

The BASIC function code is 10.

```
procedure COPRO
DIM status       : INTEGER
DIM bcd_buff(2)  : BYTE
DIM bcd2         : INTEGER
.
.
.
rem * DTL_GET_3BCD
RUN AB_BAS (10,status,ADDR(bcd_buff(1)),ADDR(bcd2))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_4BCD(); DTL_GET_WORD();
DTL_PUT_FLT(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_GET_4BCD

Gets a 4-digit BCD value from a byte array.

C Syntax

```
#include <copro.h>

unsigned DTL_GET_4BCD (in_buf,out_val)
     unsigned char *in_buf;
     unsigned *out_val;
```

Parameters

`in_buf`

Use to specify an array of two bytes that contain the 4-digit BCD value. It is assumed the data was read from a data item with a control coprocessor data type that is raw.

`out_val`

Contains the binary value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
41	DTL_E_CNVT	Data conversion error

Description

Use the DTL_GET_4BCD to convert a programmable-controller 4-digit BCD value, stored in a 2-byte array, to a control-coprocessor unsigned value.

C Example

```
unsigned status;
unsigned char thumbwheel_data;
unsigned thumbwheel_binary;
.
.
.
status = DTL_GET_4BCD (&thumbwheel_data,&thumbwheel_binary);
```

BASIC Example

The BASIC function code is 11.

```
procedure COPRO
DIM status       : INTEGER
DIM bcd_buff(2)  : BYTE
DIM bcd2         : INTEGER
.
.
.
rem * DTL_GET_4BCD
RUN AB_BAS (11,status,ADDR(bcd_buff(1)),ADDR(bcd2))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_4BCD(); DTL_PUT_WORD();
DTL_PUT_FLT(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_INIT

Creates and initializes the DTL data-definition table.

C Syntax

```
#include <copro.h>

unsigned DTL_INIT (max_defines)
     unsigned max_defines;
```

Parameters

`max_defines`

Specifies the maximum number of entries in the data-definition table. One entry is needed for each data item to be defined.

Important: Once you create the DTL data definition table, you cannot change its size within the current process.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
17	DTL_E_NO_MEM	Not enough memory available
39	DTL_NO_REINIT	DTL system already initialized

Description

Use the DTL_INIT function to initialize the data-table library before using any DTL_ function calls.

Initializes internal data and creates a data-definition table by increasing the memory area for the calling task .

Memory for the data-definition table is allocated dynamically when DTL_INIT is called. Therefore, the maximum possible size of a given task's data-definition table depends on the amount of memory available in the system's free memory pool. A call to DTL_INIT will allocate approximately 150 bytes per definition from the free-memory pool.

C Example

```
unsigned status;
status = DTL_INIT (100);         /*creates room for 100
                                DTL data definitions*/
```

BASIC Example

The BASIC function code is 1.

```
procedure COPRO
DIM status : INTEGER
.
.
.
rem * DTL_INIT - Initialize DTL for 100 definitions
RUN AB_BAS (1,status,100)
.
.
.
```

References

DTL_C_DEFINE();

DTL_PUT_FLT

Puts a floating point value into a byte array. You can use this array to write to a data item whose PLC data type is FLOAT and whose coprocessor data type is RAW.

C Syntax

```
#include <copro.h>

unsigned DTL_PUT_FLT (in_val, out_buf)
     float in_val;
     unsigned char *out_buf;
```

Parameters

`in_val`

The control-coprocessor floating-point value.

`out_buf`

Specifies an array of four bytes that will receive the floating-point value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
41	DTL_CNVT	Data conversion error (BASIC only)

Description

Use the DTL_PUT_FLT to convert a control-coprocessor float to a 4-byte array in IEEE 32-bit binary format and place it into the byte array.

C Example

```
unsigned status;
unsigned char untyped_data[50];
float flt_val;
.
.
.
status = DTL_PUT_FLT (flt_val, &untyped_data[10]);
```

BASIC Example

Important: For BASIC, the data type for the float_val parameter is REAL.

The BASIC function code is 13.

```
procedure COPRO
DIM status       : INTEGER
DIM floatbuff(4) : BYTE
DIM float_val    : REAL
.
.
.
rem * DTL_PUT_FLT
RUN AB_BAS (13, status, float_val, ADDR(floatbuff(1)))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_3BCD(); DTL_GET_4BCD();
DTL_PUT_WORD(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_PUT_WORD

Puts a word into raw format.

C Syntax

```
#include <copro.h>

unsigned DTL_PUT_WORD (in_val, out_buf)
     unsigned char in_val;
     unsigned *out_buf;
```

Parameters

`in_val`

The word value to be encoded into the byte array.

`out_buf`

Specifies an array of two bytes to receive the converted value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful

Description

Use the DTL_PUT_WORD to convert a control-coprocessor unsigned to a 2-byte array (in programmable-controller format) and place it in the 2-byte array.

C Example

```
unsigned status;
unsigned char untyped_data[50];
unsigned word_val;
.
.
.
status = DTL_PUT_WORD (word_val, &untyped_data[10]);
```

BASIC Example

The BASIC function code is 12.

```
procedure COPRO
DIM status      : INTEGER
DIM word_val    : INTEGER
DIM putbuff(2)  : BYTE
.
.
.
word_val := $ABCD
rem * DTL_PUT_WORD
RUN AB_BAS (12, status, word_val, ADDR(putbuff(1)))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_3BCD(); DTL_GET_4BCD();
DTL_PUT_FLT(); DTL_PUT_3BCD(); DTL_PUT_4BCD();

DTL_PUT_3BCD

Puts a 3-digit BCD value into a byte array.

C Syntax

```
#include <copro.h>

unsigned DTL_PUT_3BCD (in_val, out_buf)
     unsigned in_val;
     unsigned char *out_buf;
```

Parameters

`in_val`

The word value to be encoded into the byte array.

`out_buf`

Specifies an array of two bytes that will receive the converted 3-digit BCD value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
41	DTL_E_CNVT	Data-conversion error

Description

Use the DTL_PUT_3BCD to accept a longword integer value in coprocessor format in the range of 0 to 999. It converts control-coprocessor unsigned to 2-byte, 3-digit BCD value and places the result in the specified 2-byte array.

C Example

```
unsigned status;
unsigned char untyped_data[50];
unsigned word_val;
.
.
.
status = DTL_PUT_3BCD (word_val, &untyped_data[10]);
```

BASIC Example

The BASIC function code is 14.

```
procedure COPRO
DIM status       : INTEGER
DIM bcd_buff(2)  : BYTE
DIM bcd_val      : INTEGER
.
.
.
rem * DTL_PUT_3BCD
RUN AB_BAS (14, status, bcd_val, ADDR(bcd_buff(1)))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_3BCD(); DTL_GET_4BCD();
DTL_PUT_WORD(); DTL_PUT_FLT();
DTL_PUT_4BCD();

DTL_PUT_4BCD

Puts a 4-digit BCD value into a byte array.

C Syntax

```
#include <copro.h>

unsigned DTL_PUT_4BCD (in_val, out_buf)
     unsigned in_val;
     unsigned char *out_buf;
```

Parameters

`in_val`

The word value to be encoded into the byte array.

`out_buf`

Specifies an array of two bytes that will receive the converted 4-digit BCD value.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
41	DTL_E_CNVT	Data-conversion error

Description

Use the DTL_PUT_4BCD to accept a longword integer value in control-coprocessor format in the range of 0 to 9999. It converts the control-coprocessor unsigned to a 2-byte, 4-digit BCD value and places the result in the specified 2-byte array.

C Example

```
unsigned status;
unsigned char untyped_data[50];
unsigned word_val;
.
.
.
status = DTL_PUT_4BCD (word_val, &untyped_data[10]);
```

BASIC Example

The BASIC function code is 15.

```
procedure COPRO
DIM status       : INTEGER
DIM bcd_buff(2)  : BYTE
DIM bcd_val      : INTEGER
.
.
.
rem * DTL_PUT_4BCD
RUN AB_BAS (15, status, bcd_val, ADDR(bcd_buff(1)))
.
.
.
```

References

DTL_GET_WORD(); DTL_GET_FLT();
DTL_GET_3BCD(); DTL_GET_4BCD();
DTL_PUT_WORD(); DTL_PUT_FLT();
DTL_PUT_3BCD();

DTL_READ_W

Reads data from the PLC-5 programmable-controller data table to the control-coprocessor memory.

C Syntax

```
#include <copro.h>

unsigned DTL_READ_W (name_id, variable, iostat)
     unsigned name_id;
     void *variable;
     unsigned *iostat;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data item to be read is defined.

`variable`

Address of a buffer that stores the data read from the data item. Ensure the declared variable is the right type to match the data size that was specified in DTL_C_DEFINE.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
41	DTL_E_CNVT = data-conversion error
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
32	DTL_E_NODEF	No such data item defined
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_READ_W function to read data from a PLC-5 programmable controller that is directly connected to the control coprocessor.

This function is synchronous. When this function is initiated, your C application programs stops until the function completes or fails.

C Example

```
unsigned status;
unsigned machine1;
unsigned short parts1 [10];
unsigned iostat;
DTL_C_DEFINE (&machine1, “N20:36, 10, WORD, READ”);
status = DTL_READ_W (machine1, &parts1, &iostat)
if (status == DTL_SUCCESS)
{
     printf (“parts = %d\n”, parts1 [0]);
}
else
{
    (printf (“error %d, %d on read of parts data\n”,
    status,
    iostat);
}
```

BASIC Example

The BASIC function code is 5.

```
procedure COPRO
DIM status      : INTEGER
DIM fred        : INTEGER
DIM rcvbuff(10) : INTEGER
DIM iostat      : INTEGER
.
.
.
rem * DTL_READ_W - Read from N10:2 10 words into rcvbuff
RUN AB_BAS (5, status, fred, ADDR(rcvbuff), ADDR(iostat))
.
.
.
```

References

DTL_C_DEFINE(); DTL_WRITE_W();

DTL_READ_W_IDX

Reads any elements of a file, one element at a time, from the PLC-5 programmable controller to the control-coprocessor memory using only one data definition.

Important: To use this function call, you must have the versions of the ABLIB.L and CORPRO.H files that accompany Series A Revision D (1.20) or later of the Program Development Software. Contact Allen-Bradley Global Technical Support Services at (216) 646-6800 if you need these updates.

C Syntax

```
#include <copro.h>

unsigned DTL_READ_W_IDX (name_id, variable, iostat, index)
     unsigned name_id;
     void *variable;
     unsigned *iostat;
     unsigned index;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data file to be read is defined.

`variable`

Address of a buffer that stores the data read from the file. Ensure that the declared variable is the right type to match the data size that was specified in DTL_C_DEFINE.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
41	DTL_E_CNVT = data-conversion error
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

`index`

This parameter specifies the element or structure level of the data-file item to be read.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
32	DTL_E_NODEF	No such data item defined
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_READ_W_IDX function to read a file, one element at a time, from a PLC-5 programmable controller that is directly connected to the control coprocessor.

This function is synchronous. When this function is initiated, your C application programs stops until the function completes or fails.

For this function to be successful, the data definition must specify the address to the first element of the file and the number of data items must be 1.

You can address structured data types to either the structure level or the element level. When you address to the structure level, the data type must be RAW.

Valid Definition Examples

```
DTL_C_DEFINE (&idl, “N34:0”)        /* specified to element 0,
                                    default 1 item */

DTL_C_DEFINE (&idl, “T4:0.pre”)     /* index 0 accesses T4:0.pre; index 14
                                    accesses T4:14.pre */

DTL_C_DEFINE (&idl, “T4:0,1,raw”)   /* index 0 accesses all three elements of
                                    T4:0 (control, preset, accumulator);
                                    index 14 access all three elements
                                    of T4:14 (control, preset, accumulator)*/
```

Invalid Definition Example

```
DTL_C_DEFINE (&idl, “N34:3”)        /* not specified to element 0 */

DTL_C_DEFINE (&idl, “N34:0,3,long”) /* number of items not 1 */
```

C Example

```
unsigned machine;
unsigned short parts[10];
unsigned iostat;

DTL_C_DEFINE (&machine, “N20:0, 1, WORD, MODIFY”);
DTL_READ_W_IDX (machine, &parts[3], &iostat, 3)  /* read element N20:3 */
DTL_READ_W_IDX (machine, &parts[8], &iostat, 8)  /* read element N20:8 */
```

BASIC Example

The BASIC function code is 20.

```
procedure COPRO
DIM status      : INTEGER
DIM id          : INTEGER
DIM iostat      : INTEGER
DIM val3        : INTEGER
DIM val8        : INTEGER
.
.
.
rem * Define the data file
RUN AB_BAS (2, status, ADDR(id), “N10:0, 1, LONG, MODIFY”)
rem * Read N10:3 to val3
RUN AB_BAS (20, status, id, ADDR(val3), ADDR(iostat), 3)
rem * Read N10:8 to val8
RUN AB_BAS (20, status, id, ADDR(val8), ADDR(iostat), 8)
.
.
.
```

References

DTL_C_DEFINE(); DTL_WRITE_W_IDX();

DTL_RMW_W

Initiates an operation that reads a data element, modifies some of the bits, then writes it back.

C Syntax

```
#include <copro.h>

unsigned DTL_RMW_W (name_id, and_mask, or_mask, iostat)
    unsigned name_id;
    unsigned and_mask;
    unsigned or_mask;
    unsigned *iostat;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data item to be read and modified is defined.

`and_mask`

Use the and_mask to specify the bits you want to preserve in the data item. A “1” bit in the AND mask preserves the corresponding bit in the data item; a “0” bit forces the corresponding bit to zero.

`or_mask`

Use or_mask to specify the bits you want to set in the data item. A “1” bit in the OR mask forces the corresponding bit in the data item; a “0” bit forces the corresponding bit unchanged. The OR mask is applied after the AND mask.

`iostat`

Returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
41	DTL_E_CNVT = data-conversion error
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
15	DTL_E_R_ONLY	Data item defined as READ only
16	DTL_E_INVTYPE	Data is invalid type for operation
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
31	DTL_E_TOOBIG	Data item greater than maximum allowed
32	DTL_E_NODEF	No such data item defined
41	DTL_E_CNVT	Data-conversion error, I/O not attempted
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_RMW_W function to perform a read/modify/write function on a data item. The function reads a data value, modifies the data with the AND mask and then with the OR mask, and writes the data back to the programmable controller.

This synchronous function cannot be used on multiple-element data definition. The element must be a word value.

C Example

```
/*
 * Suppose there is a 16-bit “status word” in binary file 10, word 1,
 * describing the current status of the machine.  Bits 0 through 3 of
 * this word contain a code for the “current operating mode” (0-F) of
 * the machine.
*/
#define OPER_MODE_MASK 0xFFF0  /* last 4 bits = mode */
#define MANUAL_MODE 0x0002     /* bit 1*/
unsigned status;
unsigned data_id;
unsigned iostat;
.
.
.
status = DTL_C_DEFINE (&data_id, “B10:1, 1, WORD, MODIFY”);
status = DTL_RMW_W (data_id, OPER_MODE_MASK, MANUAL_MODE, &iostat);
if (status != DTL_SUCCESS)
   {
    printf (“Error %d  %d  changing to MANUAL\n”, status, iostat);
    exit (status);
   }
.
.
.
```

BASIC Example

The BASIC function code is 7.

```
procedure COPRO
DIM status      : INTEGER
DIM n7_name     : INTEGER
DIM iostat      : INTEGER
DIM and_mask     : INTEGER
DIM or_mask      : INTEGER
.
.
.
rem * DTL_RMW_W - Read/modify/write from N7:0 1 word into rcvbuff
and_mask := $2
or_mask  := $1230
RUN AB_BAS (7, status, n7_name, and_mask, or_mask, ADDR(iostat))
.
.
.
```

References

DTL_C_DEFINE(); DTL_READ_W();
DTL_WRITE_W();

DTL_RMW_W_IDX

Initiates an operation that reads a data element of the PLC-5 programmable controller, modifies some of the bits based on mask values, then writes the data element back. This function can read and modify any elements of the file using only one data definition.

Important: To use this function call, you must have the versions of the ABLIB.L and CORPRO.H files that accompany Series A Revision D (1.20) or later of the Program Development Software. Contact Allen-Bradley Global Technical Support Services at (216) 646-6800 if you need these updates.

C Syntax

```
#include <copro.h>
unsigned DTL_RMW_W_IDX (name_id, and_mask, or_mask, iostat, index)
     unsigned name_id;
     unsigned and_mask;
     unsigned or_mask;
     unsigned *iostat;
     unsigned index;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data file to be read and modified is defined.

`and_mask`

Use `and_mask` to specify the bits that you want to preserve in the data. A “1” bit in the AND mask preserves the corresponding bit in the data; a “0” bit forces the corresponding bit to zero.

`or_mask`

Use `or_mask` to specify the bits that you want to set in the data. A “1” bit in the OR mask forces the corresponding bit in the data; a “0” bit forces the corresponding bit unchanged. The OR mask is applied after the AND mask.

`iostat`

Returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
41	DTL_E_CNVT = data-conversion error
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

`index`

This parameter specifies the element or structure level of the data-file item to be read and modified.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
15	DTL_E_R_ONLY	Data item defined as READ only
16	DTL_E_INVTYPE	Data is invalid type for operation
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
31	DTL_E_TOOBIG	Data item greater than maximum allowed
32	DTL_E_NODEF	No such data item defined
41	DTL_E_CNVT	Data-conversion error, I/O not attempted
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_RMW_W_IDX function to perform a read/modify/write function on a data item. This function read a data value, modifies the data with the AND mask and then with the OR mask, and writes the data back to the programmable controller. This function allows you to read and modify any element of the file using only one data definition by specifying an index to the element.

For this function to be successful, the data definition must specify the address to the first element of the file and the number of data items must be 1.

You cannot use this synchronous function on multiple-element data definitions. The element must be a word value.

Valid Definition Examples

```
DTL_C_DEFINE (&idl, “N34:0”)        /* specified to element 0,
                                    default 1 item */
```

Invalid Definition Example

```
DTL_C_DEFINE (&idl, “N34:3”)        /* not specified to element 0 */
DTL_C_DEFINE (&idl, “N34:0,3,long”) /* number of items not 1 */
```

C Example

```
/*
 * Suppose there are 5 16-bit “status words” in binary file 10, elements
 * 0 through 4, each describing the current status of 5 machines.  Bits
 * 0 through 3 of each word contain a code for the “current mode” (0-F)
 * of the machine.  This example changes the “current mode” to a value
 * of 2 without modifying bits 5-31 for machines 0 and 4.
*/
#define MODE_AND_MASK 0xFFF0   /* preserve bits 5-31 */
#define MODE_OR_MASK 0x0002    /* set bit 1*/
#define MAC_0 0x0000           /* machine 0 index */
#define MAC_2 0x0004           /* machine 4 index */

unsigned id;
unsigned iostat;

DTL_C_DEFINE (&id, “B10:0, 1, WORD, MODIFY”);
DTL_RMW_W_IDX (id, MODE_AND_MASK, MODE_OR_MASK, &iostat, MAC_0);
DTL_RMW_W_IDX (id, MODE_AND_MASK, MODE_OR_MASK, &iostat, MAC_4);
```

BASIC Example

The BASIC function code is 22.

```
procedure COPRO
DIM status      : INTEGER
DIM id          : INTEGER
DIM iostat      : INTEGER
DIM and_mask    : INTEGER
DIM or_mask     : INTEGER

rem * Define the data file
RUN AB_BAS (2, status, ADDR(id), “B10;), 1, LONG, MODIFY”)
and_mask := $FFF0
or_mask  := $2
RUN AB_BAS (22, status, id, and_mask, or_mask, ADDR(iostat), 0))
RUN AB_BAS (22, status, id, and_mask, or_mask, ADDR(iostat), 4))
```

References

DTL_C_DEFINE(); DTL_READ_W_IDX();
DTL_WRITE_W_IDX();

DTL_SIZE

Gets the size of memory necessary to store the contents of a data item in control coprocessor format.

C Syntax

```
#include <copro.h>

unsigned DTL_SIZE (name_id, size)
     unsigned name_id;
     unsigned *size;
```

Parameters

`name_id`

The handle returned by DTL_C_DEFINE when the data item was defined.

`size`

Size (in bytes) required for the data item that is returned. Zero is returned if the data item is undefined.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NOINIT	Definition table not initialized
20	DTL_E_BADID	Definition ID out of range
32	DTL_E_NODEF	No such data item defined

Description

Use DTL_SIZE to determine the amount of control-coprocessor memory necessary to store a copy of the previously defined block of data.

C Example

```
unsigned status;
unsigned size;
unsigned integer_file;
short *integer_data;
.
.
.
status = DTL_SIZE (integer_file, &size);
if (status != DTL_SUCCESS) return (status);
integer_data = (short *) malloc (size);
.
.
.
```

BASIC Example

The BASIC function code is 16.

```
procedure COPRO
DIM status       : INTEGER
DIM n7_name      : INTEGER
DIM dtlsize      : INTEGER
.
.
.
rem * DTL_SIZE
RUN AB_BAS (16, status, n7_name, ADDR(dtlsize))
.
.
.
```

References

DTL_C_DEFINE();

DTL_TYPE

Gets the control-coprocessor data type of the named data.

C Syntax

```
#include <copro.h>

unsigned DTL_TYPE (name_id, type)
     int name_id;
     int *type;
```

Parameters

`name_id`

The handle returned by DTL_C_DEFINE when the data item was defined.

`type`

The coded value denoting the control-coprocessor data type you specified with DTL_C_DEFINE. On return from DTL_TYPE, the type variable will have one of the following values:

Type:①	Is:	Type:①	Is:
DTL_TYP_RAW	no conversion	DTL_TYP_LONG	long (int)
DTL_TYP_BYTE	char	DTL_TYP_ULONG	unsigned
DTL_TYP_UBYTE	unsigned char	DTL_TYP_FLOAT	float
DTL_TYP_WORD	short	DTL_TYP_DOUBLE	double
DTL_TYP_UWORD	unsigned short		

① These symbolic names are in COPRO.H.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NOINIT	Definition table not initialized
20	DTL_E_BADID	Definition ID out of range
32	DTL_E_NODEF	No such data item defined

Description

Use DTL_TYPE to get the code that indicates the data type you specified when you defined the data entry with DTL_C_DEFINE.

C Example

```
unsigned status;
int counter_id;
int data_type;
.
.
.
status = DTL_TYPE (counter_id, &data_type);
.
.
.
```

BASIC Example

The BASIC function code is 17.

```
procedure COPRO
DIM status       : INTEGER
DIM n7_name      : INTEGER
DIM dtltype      : INTEGER
.
.
.
rem * DTL_TYPE
RUN AB_BAS (17, status, n7_name, ADDR(dtltype))
.
.
.
```

References

DTL_C_DEFINE();

DTL_UNDEF

Deletes a data definition from the DTL data-definition table.

C Syntax

```
#include <copro.h>

unsigned DTL_UNDEF (name_id)
     unsigned name_id;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data item is defined.

Description

Use the DTL_UNDEF function to delete an entry in the data-definition table.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
19	DTL_E_NOINIT	Definition table not initialized
20	DTL_E_BADID	Definition ID out of range
32	DTL_E_NODEF	No such data item defined

C Example

```
unsigned status;
unsigned analog;
.
.
.
DTL_C_DEFINE (&analog, . . .)
.
.
.
status = DTL_UNDEF (analog);
```

BASIC Example

The BASIC function code is 3.

```
procedure COPRO
DIM status    : INTEGER
DIM analog    : INTEGER
.
.
.
rem * DTL_UNDEF Undefine item
RUN AB_BAS (3, status, analog)
.
.
.
```

References

DTL_C_DEFINE(); DTL_DEF_AVAIL();

DTL_WRITE_W

Writes data from the control-coprocessor memory to the PLC-5 programmable -controller data table.

C Syntax

```
#include <copro.h>

unsigned DTL_WRITE_W (name_id, variable, iostat)
     unsigned name_id;
     void *variable;
     unsigned *iostat;
```

Parameters

`name_id`

DTL_C_ DEFINE returns this handle when the data item to be written was defined.

`variable`

The address of a buffer that contains the data to be written to the PLC-5 programmable controller. Ensure the declared variable is the right type to match the data size that was specified in DTL_C_DEFINE.

`iostat`

Returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
15	DTL_E_R_ONLY	Data item defined as READ only
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
32	DTL_E_NODEF	No such data item defined
41	DTL_E_CNVT	Data-conversion error, I/O not attempted
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_WRITE_W function to write data to the PLC-5 programmable controller directly connected to the control coprocessor.

This function is synchronous. When this function is initiated, your C application program stops until the function completes or fails.

C Example

```
unsigned status;
unsigned machine1;
unsigned short parts1;
unsigned iostat;
.
.
.
DTL_C_DEFINE (&machine1, “N30:0, 1, WORD, MODIFY”);
.
.
.
status = DTL_WRITE_W (machine1, &parts1, &iostat)
if (status == DTL_SUCCESS)
{
     printf (“parts = %d\n”, parts1);
}
else
{
     (printf (“error %d, %d on read of parts data\n”,
              status,
              iostat);
}
.
.
.
```

BASIC Example

The BASIC function code is 6.

```
procedure COPRO
DIM status      : INTEGER
DIM fred        : INTEGER
DIM rcvbuff(10) : INTEGER
DIM iostat      : INTEGER
.
.
.
rem * DTL_WRITE_W - Write from rcvbuff 10 words to N10:2
RUN AB_BAS (6, status, fred, ADDR(rcvbuff), ADDR(iostat))
.
.
.
```

References

DTL_C_DEFINE(); DTL_READ_W(); DTL_RMW_W();

DTL_WRITE_W_IDX

Writes to any elements of a file, one element at a time, from the control-coprocessor memory to the PLC-5 programmable controller using only one data definition.

Important: To use this function call, you must have the versions of the ABLIB.L and CORPRO.H files that accompany Series A Revision D (1.20) or later of the Program Development Software. Contact Allen-Bradley Global Technical Support Services at (216) 646-6800 if you need these updates.

C Syntax

```
#include <copro.h>

unsigned DTL_WRITE_W_IDX (name_id, variable, iostat, index)
     unsigned name_id;
     void *variable;
     unsigned *iostat;
     unsigned index;
```

Parameters

`name_id`

DTL_C_DEFINE returns this handle when the data file to be written is defined.

`variable`

The address of a buffer that contains the data to be written to the PLC-5 programmable controller. Ensure that the declared variable is the right type to match the data size that was specified in DTL_C_DEFINE.

`iostat`

Returns a final completion status. Possible completion status values are:

Value	Meaning
0	DTL_SUCCESS = operation completed successfully
27	DTL_E_NOATMPT = I/O operation not attempted; see status variable for reason
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

`index`

This parameter specifies the element or structure level of the data-file item to be written.

Returns

Status	Symbolic Name	Meaning
0	DTL_SUCCESS	Operation successful
15	DTL_E_R_ONLY	Data item defined as READ only
19	DTL_E_NOINIT	DEFINE table not initialized
20	DTL_E_BADID	Definition ID out of range
24	DTL_E_FAIL	I/O completed with errors
32	DTL_E_NODEF	No such data item defined
41	DTL_E_CNVT	Data-conversion error, I/O not attempted
157	CC_E_NOTCONNECT	PLC is not connected or offline

Description

Use the DTL_WRITE_W_IDX function to write a file to the PLC-5 programmable controller connected directly to the control coprocessor.

This function is synchronous. When this function is initiated, your C application program stops until the function completes or fails.

To use this function, the data definition must specify the address to the first element of the file and the number of data items must be 1.

You can address structured data types to either the structure level or the element level. When you address to the structure level, the data type must be RAW.

Valid Definition Examples

```
DTL_C_DEFINE (&idl, “N34:0”)        /* specified to element 0,
                                    default 1 item */

DTL_C_DEFINE (&idl, “T4:0.pre”)     /* index 0 accesses T4:0.pre; index 14
                                    accesses T4:14.pre */

DTL_C_DEFINE (&idl, “T4:0,1,raw”)   /* index 0 accesses all three elements of
                                    T4:0 (control, preset, accumulator);
                                    index 14 access all three elements
                                    of T4:14 (control, preset, accumulator)*/
```

Invalid Definition Example

```
DTL_C_DEFINE (&idl, “N34:3”)        /* not specified to element 0 */
DTL_C_DEFINE (&idl, “N34:0,3,long”) /* number of items not 1 */
```

C Example

```
unsigned machine;
unsigned short parts[10];
unsigned iostat;

DTL_C_DEFINE (&machine, “N20:0, 1, WORD, MODIFY”);
DTL_WRITE_W_IDX (machine, &parts[3], &iostat, 3)  /* read element N20:3 */
DTL_WRITE_W_IDX (machine, &parts[8], &iostat, 8)  /* read element N20:8 */
```

BASIC Example

The BASIC function code is 21.

```
procedure COPRO
DIM status      : INTEGER
DIM id          : INTEGER
DIM iostat      : INTEGER
DIM val3        : INTEGER
DIM val8        : INTEGER
.
.
.
rem * Define the data file
RUN AB_BAS (2, status, ADDR(id), “N10:0, 1, LONG, MODIFY”)
rem * Write val3 to N10:3
RUN AB_BAS (21, status, id, ADDR(val3), ADDR(iostat), 3)
rem * Write val8 to N10:8
RUN AB_BAS (21, status, id, ADDR(val8), ADDR(iostat), 8)
.
.
.
```

References

DTL_C_DEFINE(); DTL_READ_W_IDX(); DTL_RMW_W_IDX();

MSG_CLR_MASK

Clears the bit associated with the specified message number.

C Syntax

```
#include <copro.h>

unsigned MSG_CLR_MASK (mask,msg_num)
     unsigned *mask;
     unsigned msg_num;
```

Parameters

`mask`

Address of the read or write mask used with the MSG_WAIT function. This function will reset the bit corresponding to the message number.

`msg_num`

Number of the PLC-5 programmable controller message (0-31).

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
133	CC_E_BAD_MSGID	Message number invalid

Description

Use the MSG_CLR_MASK function to clear bits in the message read/write masks.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Example

The BASIC function code is 44.

```
DIM status            : INTEGER
DIM msg_w_mask        : INTEGER
.
.
.
rem * MSG_CLR_MASK - clear bit in msg_w_mask for message 0
RUN AB_BAS (44,status,ADDR(msg_w_mask),0)
.
.
.
```

References

MSG_READ_HANDLER(); MSG_WAIT();
MSG_WRITE_HANDLER();

MSG_READ_HANDLER

Handles a PLC-5 programmable-controller message-read instruction.

C Syntax

```
#include <copro.h>

unsigned MSG_READ_HANDLER (variable,buff_size,msg_num,
                    items,timeout, CC_type, plc_type, iostat)
    short *variable;
    unsigned buf_size;
    unsigned msg_num;
    unsigned items;
    unsigned timeout;
    unsigned cc_type;
    unsigned plc_type;
    int *iostat;
```

Parameters

`variable`

Address of a buffer that has the data to be read.

`buff_size`

Size of the read buffer in bytes.

`msg_num`

Number of the PLC-5 programmable controller message (0-31).

`items`

Number of data items to be read by the PLC-5 programmable controller. The number and size of items cannot be greater than 240 bytes—e.g., maximum of 60 floating-point values of 4 bytes each = 240 bytes.

`timeout`

Timeout value in seconds. When using a value of CC_FOREVER (defined in COPRO.H), this function will keep the read handler posted until a message has been received.

`cc_type`

This is the data type of the control-coprocessor data buffer. Possible values are:

Value	Symbolic Name①	"C" Type
0	CC_RAW	no conversion
1	CC_BYTE	char
2	CC_UBYTE	unsigned char
3	CC_WORD	short
4	CC_UWORD	unsigned short
5	CC_LONG	long (int)
6	CC_ULONG	unsigned
7	CC_FLOAT	float
8	CC_DOUBLE	double

① These symbolic names are in COPRO.H.

`plc_type`

This is the data type of the PLC-5 data-table area. Possible values are:

Value	Symbolic Name①	"C" Type
3	PLC_WORD	short
7	PLC_FLOAT	float

① These symbolic names are in COPRO.H.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	CC_SUCCESS = operation completed successfully
101	CC_PENDING = I/O operation in progress
118	CC_E_TIME = operation did not complete in time
127	CC_E_NOATMPT = operation not attempted; see status value for reason
141	CC_E_CNVT = data-conversion error
182	CC_E_MSG_ABORT = message aborted by CC_MKILL
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
133	CC_E_BAD_MSGID	Message number invalid
157	CC_E_NOTCONNECT	PLC is not connected or offline
160	CC_E_INV_TO	Invalid timeout value
162	CC_E_INV_CTYPE	Invalid coprocessor data type
163	CC_E_INV_PTYPE	Invalid PLC-5 data type
181	CC_E_MSGPEND	Message already pending
182	CC_E_MSG_ABORT	Message aborted
190	CC_E_SIZE	Invalid size for operation
191	CC_E_TOOSMALL	Size of buffer too small

Description

Use the MSG_READ_HANDLER function to initiate the processing of unsolicited message read instructions from the PLC-5 programmable controller. This function puts an entry in the Message Control Table (MCT) for the requested message number (0-31). When the PLC-5 programmable controller executes that message number, data from the PLC-5 programmable controller is transferred to the specified user buffer.

This function is asynchronous. When this function is initiated, control is returned to the application. Use MSG_WAIT to monitor/complete the I/O operation. For the synchronous version of this command, see MSG_READ_W_HANDLER.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Example

The BASIC function code is 41.

```
DIM status            : INTEGER
DIM iostat            : INTEGER
DIM msgrbuf(5)        : INTEGER
.
.
.
rem * MSG_READ_HANDLER - Set up handler to allow for an asynchronous message
rem *                    read of msgrbuf.  This function will return to the
rem *                    user before completion of the I/O.  MSG_WAIT must be
rem *                    called to complete the I/O process.  Size of buffer
rem *                    is 20 bytes, message number is 0, number of items to
rem *                    read is 5, the timeout value is 6 seconds, the
rem *                    coprocessor data type is integer, the plc data type
rem *                    is short and iostat gets the I/O completion code.

RUN AB_BAS (41,status,ADDR(msgrbuf(1)),20,0,5,6,5,3,ADDR(iostat))
.
.
.
```

References

MSG_WRITE_HANDLER(); MSG_WAIT(); MSG_CLR_MASK(); MSG_SET_MASK(); MSG_TST_MASK(); MSG_ZERO_MASK();

Also see synchronous functions:
MSG_WRITE_W_HANDLER(); MSG_READ_W_HANDLER();

MSG_READ_W_HANDLER

Handles a PLC-5 programmable-controller generated message-read instruction.

C Syntax

```
#include <copro.h>

unsigned MSG_READ_W_HANDLER (variable,buff_size,msg_num,
                             items,timeout,cc_type,plc_type,
                             iostat)
    short *variable;
    unsigned buf_size;
    unsigned msg_num;
    unsigned items;
    unsigned timeout;
    unsigned cc_type;
    unsigned plc_type;
    int *iostat;
```

Parameters

`variable`

Address of a buffer that has the data to be read.

`buff_size`

Size of the read buffer in bytes.

`msg_num`

Number of the PLC-5 programmable controller message (0-31).

`items`

Number of data items to be read by the PLC-5 programmable controller. The number and size of items cannot be greater than 240 bytes—e.g., maximum of 60 floating-point values of 4 bytes each = 240 bytes.

`timeout`

Timeout value in seconds. A value of CC_FOREVER (defined in COPRO.H) will not return until the message has been processed.

`cc_type`

This is the data type of the control-coprocessor data buffer. Possible values are:

Value	Symbolic Name①	"C" Type
0	CC_RAW	no conversion
1	CC_BYTE	char
2	CC_UBYTE	unsigned char
3	CC_WORD	short
4	CC_UWORD	unsigned short
5	CC_LONG	long (int)
6	CC_ULONG	unsigned
7	CC_FLOAT	float
8	CC_DOUBLE	double

① These symbolic names are in COPRO.H.

`plc_type`

This is the data type of the PLC-5 data table area. Possible values are:

Value	Symbolic Name①	"C" Type
3	PLC_WORD	short
7	PLC_FLOAT	float

① These symbolic names are in COPRO.H.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	CC_SUCCESS = operation completed successfully
118	CC_E_TIME = operation did not complete in time
127	CC_E_NOATMPT = operation not attempted; see status value for reason
141	CC_E_CNVT = data-conversion error
182	CC_E_MSG_ABORT = message aborted by CC_MKILL
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
124	CC_E_FAIL	I/O completed with errors
133	CC_E_BAD_MSGID	Message number invalid
157	CC_E_NOTCONNECT	PLC is not connected or offline
160	CC_E_INV_TO	Invalid timeout value
162	CC_E_INV_CTYPE	Invalid coprocessor data type
163	CC_E_INV_PTYPE	Invalid PLC-5 data type
181	CC_E_MSGPEND	Message already pending
182	CC_E_MSG_ABORT	Message aborted
190	CC_E_SIZE	Invalid size for operation
191	CC_E_TOOSMALL	Size of buffer too small

Description

Use the MSG_READ_W_HANDLER function to handle unsolicited message read instructions from the PLC-5 programmable controller. This function puts an entry in the Message Control Table (MCT) for the requested message number (0-31). When the PLC-5 programmable controller executes that message number, data from the user-specified buffer is transferred to the PLC-5 programmable controller.

This function is synchronous. When this function is initiated, the application program stops until the function completes or fails. For the asynchronous version of this command see MSG_READ_HANDLER.

C Example

```
  short variable [4]              /* buffer to store read data */
  unsigned timeout = 45;          /* 45 second timeout */
  unsigned msgnum = 10;           /* plc message number 10 */
  unsigned cc_type = 3;           /* CC_WORD = 3; */
  unsigned plc_type = 3;          /* PLC_WORD = 3; */
  unsigned items = 4;             /* 4 words of data to be read */
  int iostat;                     /* iostatus return value */
  int rtn_val;                    /* function return value */
  .
  .
  .
  variable [0] = 1;
  variable [1] = 9;
  variable [2] = 9;
  variable [3] = 2;

  rtn_val = MSG_READ_W_HANDLER ( variable, sizeof (variable), msgnum, items,
                       timeout, cc_type, PLC_type, &iostat);
```

BASIC Example

The BASIC function code is 40.

```
DIM status            : INTEGER
DIM iostat            : INTEGER
DIM msgrbuf(5)        : INTEGER
.
.
.
rem * MSG_READ_W_HANDLER - Set up handler to allow for a synchronous message
rem *                      read of msgrbuf.  This function will wait for
rem *                      completion of the I/O before returning to the user.
rem *                      Size of buffer is 20 bytes, message number is 0,
rem *                      number of items to read is 5, the timeout value is
rem *                      6 seconds, the coprocessor data type is integer,
rem *                      the plc data type is short and iostat gets the I/O
rem *                      completion code.

RUN AB_BAS (40,status,ADDR(msgrbuf(1)),20,0,5,6,5,3,ADDR(iostat))
.
.
.
```

References

MSG_WRITE_W_HANDLER();

Also see asynchronous functions
MSG_READ_HANDLER(); MSG_WRITE_HANDLER();

MSG_SET_MASK

Sets the bit associated with the specified message number.

C Syntax

```
#include <copro.h>

unsigned MSG_SET_MASK (mask,msg_num)
     unsigned *mask;
     unsigned msg_num;
```

Parameters

`mask`

Address of the read or write mask used with the MSG_WAIT function. This function will set the bit corresponding to the message number.

`msg_num`

Number of the PLC-5 programmable-controller message (0-31).

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
133	CC_E_BAD_MSGID	Message number invalid

Description

Use the MSG_SET_MASK function to set bits in the message read/write masks.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Examples

The BASIC function code is 45.

```
DIM status            : INTEGER
DIM msg_w_mask        : INTEGER
.
.
.
rem * MSG_SET_MASK - set bit in msg_w_mask for message 2
RUN AB_BAS (45,status,ADDR(msg_w_mask),2)
.
.
.
```

References

MSG_READ_HANDLER(); MSG_WRITE_HANDLER(); MSG_WAIT(); MSG_CLR_MASK(); MSG_TST_MASK(); MSG_ZERO_MASK();

MSG_TST_MASK

Tests the bit associated with the specified message number.

C Syntax

```
#include <copro.h>

status = MSG_TST_MASK (mask,msg_num)
     unsigned *mask;
     unsigned msg_num;
```

Parameters

`mask`

Address of the read or write mask used with the MSG_WAIT function. This function will test the bit corresponding to the message number.

`msg_num`

Number of the PLC-5 programmable-controller message (0-31).

Returns

Value	Meaning
1	Returns 1 if the corresponding bit is set
0	Returns 0 if the corresponding bit is reset or msg_num is not 0-31

Description

Use the MSG_TST_MASK function to test bits in the message read/write masks.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Example

The BASIC function code is 46.

```
DIM status            : INTEGER
DIM msg_w_mask        : INTEGER
.
.
.
rem * MSG_TST_MASK - test bit in msg_w_mask for message 1
RUN AB_BAS (46,status,ADDR(msg_w_mask),1)
.
.
.
```

References

MSG_READ_HANDLER(); MSG_WRITE_HANDLER(); MSG_WAIT(); MSG_SET_MASK(); MSG_CLR_MASK(); MSG_ZERO_MASK();

MSG_WAIT

Wait for one or more messages to complete.

C Syntax

```
#include <copro.h>

unsigned MSG_WAIT (r_mask,w_mask,sync,
                   r_rslt_mask,w_rslt_mask)
    unsigned r_mask;
    unsigned w_mask;
    unsigned sync;
    unsigned *r_rslt_mask;
    unsigned *w_rslt_mask;
```

Parameters

Important: A bit set in the result mask indicates that the message is completed; however, it does not indicate that the operation completed without error. You must check the final completion status of each I/O operation to verify that no error occurred.

`r_mask`

Bit map of the requested read message instructions. Bit 0 = message 0; bit 31 = message 31. If the bit is set, this function checks for completion of the message.

`w_mask`

Bit map of the requested write message instructions. Bit 0 = message 0; bit 31 = message 31. If the bit is set, this function checks for completion of the message.

`sync`

Use this parameter to specify if the function will return immediately (0) if none of the requested messages have completed or if the function will wait (1) for at least one message to complete before returning to the user.

`r_rslt_mask`

Bit map of the results from the requested read message instructions. Bit 0 = message 0; bit 31 = message 31. If the bit is set, this message has completed I/O or has timed out.

`w_rslt_mask`

Bit map of the results from the requested write message instructions. Bit 0 = message 0; bit 31 = message 31. If the bit is set, this message has completed I/O or has timed out.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful

Description

Use MSG_WAIT to wait for one or more asynchronous message operations to complete. MSG_WAIT will check for the completion of any combination of pending read/write message numbers. The message numbers are encoded in a read and write mask. The corresponding message entry is checked for I/O completion. If the message has completed, the iostat entry for that message is updated and the corresponding bit in the read/write result mask is set. If none of the requested messages have completed I/O and the sync parameter is 0, this function will return immediately to the caller (asynchronous). Otherwise, this function will wait until at least one of the requested messages has completed.

Important: If both r_mask and w_mask are 0 or if none of the messages for which bits are set were requested, the calling process will wait forever (only if sync=1).

C Example

```
@_sysedit: equ 1        /* Edition 1. */
@_sysattr: equ 0x8000   /* Revision 00 (re-entrant 0x80 << 8 + revision 00) */
#include                <copro.h>
main ()
{
  int read_var,write_var,one_shot,iostat,status;
  unsigned rm,wm,ret_rm,ret_wm;
  read_var = 1;
  one_shot = 42;
  CC_INIT();                                              /* initialize copro */
  status = MSG_ZERO_MASK(&rm);             /* clear out read and write masks */
  status = MSG_ZERO_MASK(&wm);
  status = MSG_SET_MASK(&rm,0);      /* wait for read msg 0 & 1, write msg 1 */
  status = MSG_SET_MASK(&rm,1);
  status = MSG_SET_MASK(&wm,1);
                                           /* post initial message handlers */
  status = MSG_READ_HANDLER ( &one_shot, sizeof (one_shot), 1, 1, 45,
                              CC_LONG, PLC_WORD, &iostat);
  status = MSG_READ_HANDLER ( &read_var, sizeof (read_var), 0, 1, 45,
                              CC_LONG, PLC_WORD, &iostat);
```

```
  status = MSG_WRITE_HANDLER ( &write_var, sizeof (write_var), 1, 1, 45,
                               CC_LONG, PLC_WORD, &iostat);
  while (1)
  {
   status = MSG_WAIT (rm,wm,1,&ret_rm,&ret_wm);   /* wait for either message */
   if (MSG_TST_MASK (&ret_rm,1))           /* read msg 1 completed - one_shot */
         {
         printf ("One shot data was read\n");
         status = MSG_CLR_MASK(&rm,1);          /* dont wait for it any more */
         }
   if (MSG_TST_MASK (&ret_rm,0))                      /* read msg 0 completed */
         {
           printf ("Read message 0 occurred\n");
           status = MSG_READ_HANDLER ( &read_var, sizeof (read_var), 0, 1, 45,
                                       CC_LONG, PLC_WORD, &iostat);
         }
   if (MSG_TST_MASK (&ret_wm,1))                    /* write msg 1 completed */
         {
           printf ("Write message 1 occurred\n");
           status = MSG_WRITE_HANDLER ( &write_var, sizeof (write_var), 1, 1,
                                        45, CC_LONG, PLC_WORD, &iostat);
         }
    }
}
```

BASIC Example

The BASIC function code is 48.

```
 DIM status            : INTEGER
 DIM iostat            : INTEGER
 DIM rm                : INTEGER
 DIM wm                : INTEGER
 DIM ret_rm            : INTEGER
 DIM ret_wm            : INTEGER
 .
 .
 .
 rem * MSG_WAIT - wait for completion of messages based on bit pattern
 rem *            in rm (read mask) and wm (write mask).  The value 1
 rem *            indicates that this function will wait until at least
 rem *            one of the requested messages is complete.  The results
 rem *            of the wait are stored in ret_rm (read) and ret_wm (write).

 RUN AB_BAS (48,status,rm,wm,1,ADDR(ret_rm),ADDR(ret_wm))
 .
 .
 .
```

References

MSG_READ_HANDLER(); MSG_WRITE_HANDLER(); MSG_SET_MASK(); MSG_CLR_MASK(); MSG_TST_MASK(); MSG_ZERO_MASK();

MSG_WRITE_HANDLER

Handles a PLC-5 programmable-controller message-write instruction.

C Syntax

```
#include <copro.h>

unsigned MSG_WRITE_HANDLER (variable,buff_size,msg_num,
                            items,timeout,CC_type,plc_type,
                            iostat)
    short *variable;
    unsigned buf_size;
    unsigned msg_num;
    unsigned items;
    unsigned timeout;
    unsigned cc_type;
    unsigned plc_type;
    int *iostat;
```

Parameters

`variable`

Address of a buffer that stores the write data.

`buff_size`

Size of the write buffer in bytes.

`msg_num`

Number of the PLC-5 programmable-controller message (0-31).

`items`

Number of data items to be written by the PLC-5 programmable controller. The number and size of the items cannot be greater than 234 bytes—e.g., maximum of 58 floating-point values of 4 bytes each are < 234 bytes.

`timeout`

Timeout value in seconds. When using a value of CC_FOREVER (defined in COPRO.H), this function keeps the write handler posted until a message has been received.

`cc_type`

This is the data type of the control coprocessor's data buffer. Possible values are:

Value	Symbolic Name①	"C" Type
0	CC_RAW	no conversion
1	CC_BYTE	char
2	CC_UBYTE	unsigned char
3	CC_WORD	short
4	CC_UWORD	unsigned short
5	CC_LONG	long (int)
6	CC_ULONG	unsigned
7	CC_FLOAT	float
8	CC_DOUBLE	double

① These symbolic names are in COPRO.H.

`plc_type`

This is the data type of the PLC-5 programmable-controller data-table area. Possible values are:

Value	Symbolic Name①	"C" Type
3	PLC_WORD	short
7	PLC_FLOAT	float

① These symbolic names are in COPRO.H.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	CC_SUCCESS = operation completed successfully
101	CC_PENDING = I/O operation in progress
118	CC_E_TIME = operation did not complete in time
127	CC_E_NOATMPT = operation not attempted; see status value for reason
141	CC_E_CNVT = data-conversion error
182	CC_E_MSG_ABORT = message aborted by CC_MKILL
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
133	CC_E_BAD_MSGID	Message number invalid
157	CC_E_NOTCONNECT	PLC is not connected or offline
160	CC_E_INV_TO	Invalid timeout value
162	CC_E_INV_CTYPE	Invalid coprocessor data type
163	CC_E_INV_PTYPE	Invalid PLC-5 data type
181	CC_E_MSGPEND	Message already pending
182	CC_E_MSG_ABORT	Message aborted
190	CC_E_SIZE	Invalid size for operation
191	CC_E_TOOSMALL	Size of buffer too small

Description

Use the MSG_WRITE_HANDLER function to initiate the processing of unsolicited message write instructions from the PLC-5 programmable controller. This function puts an entry in the Message Control Table (MCT) for the requested message number (0-31). When the PLC-5 programmable controller executes that message number, data from the PLC-5 programmable controller is transferred to the specified user buffer.

This function is asynchronous. When this function is initiated, control is returned to the application. Use MSG_WAIT to monitor/complete the I/O operation. For the synchronous version of this command see MSG_WRITE_W_HANDLER.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Example

The BASIC function code is 43.

```
DIM status            : INTEGER
DIM iostat            : INTEGER
DIM msgwbuf(5)        : INTEGER
.
.
.
rem * MSG_WRITE_W_HANDLER - Set up handler to allow for an asynchronous message
rem *                       write of msgwbuf.  This function will return to the
rem *                       user befor completion of the I/O. MSG_WAIT must be
rem *                       called to complete the I/O process.  Size of buffer
rem *                       is 20 bytes, message number is 0, umber of items
rem *                       to read is 5, the timeout value is 6 seconds, the
rem *                       coprocessor data type is integer, the plc data type
rem *                       is short and iostat gets the I/O completion code.

RUN AB_BAS (43,status,ADDR(msgwbuf(1)),20,1,5,6,5,3,ADDR(iostat))
.
.
.
```

References

MSG_READ_HANDLER(); MSG_WAIT(); MSG_CLR_MASK();
MSG_SET_MASK(); MSG_TST_MASK(); MSG_ZERO_MASK();

Also see synchronous functions:
MSG_WRITE_W_HANDLER(); MSG_READ_W_HANDLER();

MSG_WRITE_W_HANDLER

Handles a PLC-5 programmable-controller generated message-write instruction.

C Syntax

```
#include <copro.h>

unsigned MSG_WRITE_W_HANDLER (variable,buff_size,msg_num,
                    items,timeout,cc_type,plc_type,iostat)
    short *variable;
    unsigned buff_size;
    unsigned msg_num;
    unsigned items;
    unsigned timeout;
    unsigned cc_type;
    unsigned plc_type;
    int *iostat;
```

Parameters

`variable`

Address of a buffer that stores the write data.

`buff_size`

Size of the write buffer in bytes.

`msg_num`

Number of the PLC-5 programmable-controller message (0-31).

`items`

Number of data items to be written by the PLC-5 programmable controller. The number and size of the items cannot be greater than 234 bytes—e.g., maximum of 58 floating-point values of 4 bytes each are < 234 bytes.

`timeout`

Timeout value in seconds. A value of CC_FOREVER (defined in COPRO.H) will not return until the message has been processed.

`cc_type`

This is the data type of the control-coprocessor's data buffer. Possible values are:

Value	Symbolic Name①	"C" Type
0	CC_RAW	no conversion
1	CC_BYTE	char
2	CC_UBYTE	unsigned char
3	CC_WORD	short
4	CC_UWORD	unsigned short
5	CC_LONG	long (int)
6	CC_ULONG	unsigned
7	CC_FLOAT	float
8	CC_DOUBLE	double

① These symbolic names are in COPRO.H.

`plc_type`

This is the data type of the PLC-5 programmable-controller data-table area. Possible values are:

Value	Symbolic Name①	"C" Type
3	PLC_WORD	short
7	PLC_FLOAT	float

① These symbolic names are in COPRO.H.

`iostat`

This parameter returns a final completion status. Possible completion status values are:

Value	Meaning
0	CC_SUCCESS = operation completed successfully
118	CC_E_TIME = operation did not complete in time
127	CC_E_NOATMPT = operation not attempted; see status value for reason
141	CC_E_CNVT = data-conversion error
182	CC_E_MSG_ABORT = message aborted by CC_MKILL
*xxx*①	PCCC_E_*xxx* = operation refused by the PLC-5 programmable controller

① See Table B.A for PCCC errors.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
124	CC_E_FAIL	I/O completed with errors
133	CC_E_BAD_MSGID	Message ID out of range
157	CC_E_NOTCONNECT	PLC is not connected or offline
160	CC_E_INV_TO	Invalid timeout value
162	CC_E_INV_CTYPE	Invalid coprocessor data type
163	CC_E_INV_PTYPE	Invalid PLC-5 data type
181	CC_E_MSGPEND	Message already pending
182	CC_E_MSG_ABORT	Message aborted
190	CC_E_SIZE	Invalid size for operation
191	CC_E_TOOSMALL	Size of buffer too small

Description

Use the MSG_WRITE_W_HANDLER function to initiate the processing of unsolicited message-write instructions from the PLC-5 programmable controller. This function puts an entry in the Message Control Table (MCT) for the requested message number (0-31). When the PLC-5 programmable controller executes that message number, data from the PLC-5 programmable controller is transferred to the specified user buffer.

This function is synchronous. When this function is initiated, the application program stops until the function completes or fails. For the asynchronous version of this command, see MSG_WRITE_HANDLER.

C Example

```
.
.
.
short variable [4]              /* buffer to receive write data */
unsigned timeout = 45;          /* 45 second timeout */
unsigned msgnum = 10;           /* plc message number 10 */
unsigned cc_type = 3;           /* CC_WORD = 3; */
unsigned plc_type = 3;          /* PLC_WORD = 3; */
unsigned items = 4;             /* 4 words of data to be written by PLC-5 */
int iostat;                     /* iostatus return value */
int rtn_val;                    /* function return value */

rtn_val = MSG_WRITE_W_HANDLER (&variable, sizeof (variable), msgnum, items,
                    timeout, cc_type, PLC_type, &iostat);
.
.
.
```

BASIC Example

The BASIC function code is 42.

```
DIM status            : INTEGER
DIM iostat            : INTEGER
DIM msgwbuf(5)        : INTEGER
.
.
.
rem * MSG_WRITE_W_HANDLER - Set up handler to allow for a synchronous message
rem *                       write of msgwbuf.  This function will wait for
rem *                       completion of the I/O before returning to the user.
rem *                       Size of buffer is 20 bytes, message number is 0,
rem *                       number of items to read is 5, the timeout value is
rem *                       6 seconds, the coprocessor data type is integer,
rem *                       the plc data type is short and iostat gets the I/O
rem *                       completion code.

RUN AB_BAS (42,status,ADDR(msgwbuf(1)),20,1,5,6,5,3,ADDR(iostat))
.
.
.
```

References

MSG_READ_W_HANDLER();

Also see asynchronous functions
MSG_READ_HANDLER(); MSG_WRITE_HANDLER();

MSG_ZERO_MASK

Zeros all bits in the specified mask.

C Syntax

```
#include <copro.h>

unsigned MSG_ZERO_MASK (mask)
     unsigned *mask;
```

Parameters

`mask`

This is the address of the read or write mask used with the MSG_WAIT function. This function will reset all bits in the mask.

Returns

Status	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful

Description

Use the MSG_ZERO_MASK function to zero bits in the message read/write masks.

C Example

See MSG_WAIT on page B-102 for a complete example of asynchronous message processing.

BASIC Example

The BASIC function code is 47.

```
DIM status            : INTEGER
DIM msg_r_mask        : INTEGER
.
.
.
rem * MSG_ZERO_MASK - zero out msg_r_mask
RUN AB_BAS (47,status,ADDR(msg_r_mask))
.
.
.
```

References

MSG_READ_HANDLER(); MSG_WRITE_HANDLER(); MSG_WAIT(); MSG_SET_MASK(); MSG_CLR_MASK(); MSG_TST_MASK();

TAG_DEF_AVAIL

Returns the number of TAG definitions available in the TAG table.

C Syntax

```
#include <copro.h>

unsigned TAG_DEF_AVAIL ( )
```

Parameters

Returns

Value	Meaning
xx	The number of TAG definitions available in the TAG table

Description

Use TAG_DEF_AVAIL to determine the number of TAG definitions available in the system's TAG table. The function calculates the difference between the number of entries defined in the online utility and the number of successful TAG definitions made using TAG_DEFINE.

C Example

```
unsigned avail_val;
avail_val = TAG_DEF_AVAIL ( );
```

BASIC Example

The BASIC function code is 63.

References

TAG_DEFINE(); TAG_UNDEF(); TAG_LINK();
TAG_GLOBAL_UNDEF();

TAG_DEFINE

Adds an entry to the control-coprocessor TAG table.

C Syntax

```
#include <copro.h>

unsigned TAG_DEFINE (name_id,tag_addr,
                     tag_name,tag_size,access)
    unsigned *name_id;
    unsigned tag_addr;
    unsigned char *tag_name;
    unsigned tag_size;
    unsigned char access;
```

Parameters

`name_id`

Name_id is used to return a handle assigned by the TAG library to the TAG name.

`tag_addr`

Pointer to the start of the user's tagged area.

`tag_name`

Specifies the name of the user's TAG. This is a null-terminated ASCII string of up to 9 characters. The TAG name can contain the following characters: A-Z, a-z, 0-9, and _. The first character must be alphabetic.

`tag_size`

Specifies the number of consecutive bytes starting at the tag_addr to be included in the tagged area. This number must be ≤ 240.

`access`

Specifies either READ or READ/WRITE access to the tagged area by other processes. Possible values are:

Value	Symbolic Name①	Access
0	TG_READ	Only the process that created the TAG can modify it; other processes can only READ the TAG
1	TG_MODIFY	Any process can READ or modify the TAG

① These symbols are in COPRO.H.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
131	CC_E_TOOBIG	Data item is greater than maximum allowed
175	CC_E_BADTAG	Invalid TAG name
177	CC_E_TAGFULL	TAG table is full
184	CC_E_DUP	Duplicate TAG
186	CC_E_BADACC	Bad value for TAG access

Description

Use TAG_DEFINE to place a TAG name entry into the TAG table. The TAG name is a symbolic reference to the user's designated data area. The TAG_DEFINE routine also returns a handle with which the calling task can refer to the TAG area on subsequent TAG_ calls. This handle is an offset into the TAG table. This makes subsequent access to the table faster than doing a symbolic name search.

When a process defines a TAG name, a pointer referencing the tagged memory is stored in the TAG table. If the process that defined the TAG aborts or is terminated, the memory referenced by the TAG pointer is returned to the free memory pool. The TAG pointer still exists in the TAG table, but that memory no longer addresses the TAG and may contain invalid data.

ATTENTION: A process that creates any TAG must not terminate if that TAG is to remain valid. To correctly remove a TAG, use the TAG_UNDEF or TAG_GLOBAL_UNDEF function or reset the coprocessor module.

C Example

```
unsigned off;
unsigned fred;
unsigned status;
status = TAG_DEFINE
(&off,&fred,"Fred",sizeof(fred),TG_MODIFY);
```

BASIC Example

The BASIC function code is 60.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
DIM george            : INTEGER
.
.
.
rem * TAG_DEFINE - Define a tag to variable george with a symbolic
rem *              name George.  The size of george is 4 bytes.
RUN AB_BAS (60,status,ADDR(tag_id),ADDR(george),“George”,4,1)
.
.
.
```

References

TAG_UNDEF(); TAG_DEF_AVAIL(); TAG_LINK();
TAG_GLOBAL_UNDEF();

TAG_GLOBAL_UNDEF

Removes a TAG or TAGs from the TAG table defined by any calling process.

C Syntax

```
#include <copro.h>

unsigned TAG_GLOBAL_UNDEF (tag)
    unsigned tag;
       or
    char *tag;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call. A value of CC_ALLTAGS (defined in COPRO.H) will remove all TAGs defined in the TAG table.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
176	CC_E_NOTAG	TAG not found

Description

Use the TAG_UNDEF function to remove a TAG or TAGs from the TAG table.

C Example

```
unsigned status;
status = TAG_GLOBAL_UNDEF (“Fred”);
```

BASIC Example

The BASIC function code is 62.

```
DIM status            : INTEGER
.
.
.
rem * TAG_GLOBAL_UNDEF - undefine the tag “steps” created by another process
RUN AB_BAS (62,status,“steps”)
.
.
.
```

References

TAG_DEFINE(); TAG_DEF_AVAIL(); TAG_LINK();
TAG_GLOBAL_UNDEF();

TAG_LINK

Gets the handle from TAG name.

C Syntax

```
#include <copro.h>

unsigned TAG_LINK (name_id,tag_name)
     unsigned *name_id;
     unsigned char *tag_name;
```

Parameters

`name_id`

Name_id is used to return a handle assigned by the TAG library to the TAG name.

`tag_name`

Specifies the name of the user's TAG. This is a null-terminated ASCII string of up to 9 characters. The TAG name can contain the following characters: A-Z, a-z, 0-9, and _. The first character must be alphabetic.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
176	CC_E_NOTAG	TAG not found

Description

Use the TAG_LINK function to get a TAG handle for a TAG-table entry. This handle is an offset into the TAG table. It makes subsequent access to the table faster than doing a symbolic name search.

C Example

```
unsigned status;
unsigned fred_id;
.
.
.
status = TAG_LINK (&fred_id,“Fred”);
```

BASIC Example

The BASIC function code is 64.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
.
.
.
rem * TAG_LINK - link to the tag “wall” defined by another process
RUN AB_BAS (64,status,ADDR(tag_id),“wall”)
.
.
.
```

References

TAG_DEFINE(); TAG_UNDEF(); TAG_DEF_AVAIL();
TAG_GLOBAL_UNDEF();

TAG_LOCK

This function locks the requested TAG memory area.

C Syntax

```
#include <copro.h>

unsigned TAG_LOCK (tag,timeout)
     unsigned tag;
        or
     char *tag;
     unsigned timeout;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

`timeout`

Timeout value in seconds, from 0 to 16,383. A value of CC_FOREVER (defined in COPRO.H) will wait until the TAG has been locked.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
160	CC_E_INV_TO	Invalid timeout value
171	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
176	CC_E_NOTAG	TAG not found

Description

Use the TAG_LOCK function to protect against concurrent access to the tagged data when accessing the TAG without using TAG_READ or TAG_WRITE.

Important: After access to the TAG is completed, you must call TAG_UNLOCK to unlock the TAG; otherwise, the system may hang.

A status of CC_SUCCESS indicates that the calling procedure has locked the TAG.

C Example

```
unsigned name_id;
register int x;                                /* loop counter */
unsigned status;                               /* status return */
int Fred_ptr [12]                              /* pointer to Fred data area */
int buffer [12];                               /* transfer buffer */
.
.
.
status - TAG_DEFINE (&name_id, Fred_ptr, “Fred”, 12 * sizeof (int),TG_MODIFY);
.
.
.
status = TAG_LOCK (“Fred”,30);                 /* lock TAG Fred */
if (status = CC_SUCCESS) exit (status);        /* exit if wrong TAG */
for (x=0; x<12; ++x)  *Fred_ptr = buffer [x]; /* transfer data to Fred*/
status = TAG_UNLOCK (“Fred”) ;                 /* unlock Fred */
.
.
.
/* the above example from TAG_LOCK to TAG_UNLOCK would be functionally
equivalent to the following */
status = TAG_WRITE (“Fred”, 0, 12 * sizeof (int), &buffer,30);
```

BASIC Example

The BASIC function code is 69.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
.
.
.
rem * TAG_LOCK - lock the tag specified by tag_id from concurrent access
rem *            the timeout is 30 seconds.
RUN AB_BAS (69,status,tag_id,30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF();
TAG_READ(); TAG_WRITE(); TAG_UNLOCK();

TAG_READ

Reads data from a user's TAG memory area.

C Syntax

```
#include <copro.h>

unsigned TAG_READ (tag,offset,size,buffer,timeout)
    unsigned tag;
        or
    char *tag
    unsigned offset;
    unsigned size;
    unsigned *buffer;
    unsigned timeout;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

`offset`

A byte offset from the start of the tagged area from which data will be read.

`size`

Specifies the number of bytes to read from the tagged area.

`buffer`

Specifies the buffer to copy the data read from the tagged area.

`timeout`

Timeout value in seconds (valid range 0-16383). The function will timeout unless the TAG can be read before the timeout expires. The TAG may not be able to be read if another process has the TAG locked. A value of CC_FOREVER (defined in COPRO.H) will cause this function to wait indefinitely until the TAG can be read.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
131	CC_E_TOOBIG	Data item is greater than maximum allowed
160	CC_E_INV_TO	Invalid timeout value
171	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
176	CC_E_NOTAG	TAG not found

Description

Use the TAG_READ function to read data from a tagged memory area. This function guarantees that the read data area is semaphored during the read operation.

C Example

```
unsigned my_fred;
unsigned status;
.
.
.
status = TAG_READ (“Fred”,0,sizeof(my_fred),&my_fred,30);
```

BASIC Example

The BASIC function code is 66.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
DIM my_data           : INTEGER
.
.
.
rem * TAG_READ - read 4 bytes from a tag, starting at offset 0 into
rem *            my_data with a timeout of 30 seconds.
RUN AB_BAS (66,status,tag_id,0,4,ADDR(my_data),30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF(); TAG_GLOBAL_UNDEF(); TAG_WRITE(); TAG_LOCK(); TAG_UNLOCK(); TAG_READ_W(); TAG_WRITE_W();

TAG_READ_W

Reads data from a user's TAG memory area after the TAG is written by TAG_WRITE_W.

C Syntax

```
#include <copro.h>

unsigned TAG_READ_W (tag,offset,size,buffer,timeout)
     unsigned tag;
         or
     unsigned char *tag;
     unsigned offset;
     unsigned size;
     unsigned *buffer;
     unsigned timeout;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

`offset`

A byte offset from the start of the tagged area from which data will be read.

`size`

Specifies the number of bytes to read from the tagged area.

`buffer`

Specifies the buffer to copy the data read from the tagged area.

`timeout`

Timeout value in seconds (valid range 0-16383). The function will timeout unless the TAG can be read before the timeout expires. The TAG may not be able to be read if another process has the TAG locked or if the corresponding TAG_WRITE_W from another process has not been issued. A value of CC_FOREVER (defined in COPRO.H) will cause this function to wait indefinitely until the TAG can be read.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
131	CC_E_TOOBIG	Data item is greater than maximum allowed
160	CC_E_INV_TO	Invalid timeout value
171	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
173	CC_E_TIME_NOWRITE	Did not complete in time, TAG not written
176	CC_E_NOTAG	TAG not found

Description

Use the TAG_READ_W function to read data from a tagged memory area. This function waits until a corresponding TAG_WRITE_W function is posted. More than one TAG_READ_W can be pending on a single TAG_WRITE_W. This function guarantees that the read data area is semaphored during the read operation.

C Example

```
unsigned my_fred;
unsigned status;
.
.
.
status = TAG_READ_W (“Fred”,0,sizeof(my_fred),&my_fred,30);
```

BASIC Example

The BASIC function code is 65.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
DIM my_data           : INTEGER
.
.
.
rem * TAG_READ_W - read 4 bytes from a tag, starting at offset 0 into
rem *              my_data with a timeout of 30 seconds.  The read will
rem *              not proceed until the specified tag has been written
rem *              to by TAG_WRITE_W.
RUN AB_BAS (65,status,tag_id,0,4,ADDR(my_data),30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF(); TAG_GLOBAL_UNDEF(); TAG_WRITE(); TAG_LOCK(); TAG_UNLOCK(); TAG_READ(); TAG_WRITE_W();

TAG_UNDEF

Removes a TAG or TAGs from the TAG table defined by the calling process.

C Syntax

```
#include <copro.h>

unsigned TAG_UNDEF (tag)
    unsigned tag;
        or
    char *tag;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call. A value of CC_ALLTAGS (defined in COPRO.H) will remove all TAGs defined by this process.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
176	CC_E_NOTAG	TAG not found
179	CC_E_NOTDEFINER	Caller is not the definer of this TAG

Description

Use the TAG_UNDEF function to remove a TAG or TAGs from the TAG table. This function can only remove TAGs defined by the calling process.

C Example

```
unsigned status;
status = TAG_UNDEF (“Fred”);
```

BASIC Example

The BASIC function code is 61.

```
DIM status            : INTEGER
.
.
.
rem * TAG_UNDEF - undefine the tag “Fred” created by my process
RUN AB_BAS (61,status,“Fred”)
.
.
.
```

References

TAG_DEFINE(); TAG_DEF_AVAIL(); TAG_LINK();
TAG_GLOBAL_UNDEF();

TAG_UNLOCK

This function unlocks the requested TAG memory area.

C Syntax

```
#include <copro.h>

unsigned TAG_UNLOCK (tag)
     unsigned tag
         or
     char *tag;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
176	CC_E_NOTAG	TAG not found
178	CC_E_NOTLOCKER	Caller is not the locker of this TAG
185	CC_E_NOTLOCKED	TAG is not locked

Description

Use the TAG_UNLOCK function to unlock the TAG locked by TAG_LOCK.

Important: This function must be called after access to the TAG is completed; otherwise, the system may hang.

C Example

See the TAG_LOCK() example on page B-123.

BASIC Example

The BASIC function number is 70.

```
DIM status            : INTEGER
.
.
.
rem * TAG_UNLOCK - unlock the tag “Fred”, the timeout is 30 seconds
RUN AB_BAS (70,status,“Fred”,30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF();
TAG_READ(); TAG_WRITE(); TAG_LOCK();

TAG_WRITE

Writes data to a user's TAG memory area.

C Syntax

```
#include <copro.h>

unsigned TAG_WRITE (tag,offset,size,buffer,timeout)
    unsigned tag;
        or
    char *tag
    unsigned offset;
    unsigned size;
    unsigned char *buffer;
    unsigned timeout;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

`offset`

A byte offset from the start of the tagged area from where data will be written.

`size`

Specifies the number of bytes to write to the tagged area.

`buffer`

Specifies the buffer to copy the write data write to the tagged area.

`timeout`

Timeout value in seconds (valid range 0-16383). The function will timeout unless the TAG can be written before the timeout expires. The TAG may not be able to be written if another process has the TAG locked. A value of CC_FOREVER (defined in COPRO.H) will cause this function to wait indefinitely until the TAG can be written.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
131	CC_E_TOOBIG	Data size greater than the maximum allowed
160	CC_E_INV_TO	Invalid timeout value
171	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
174	CC_E_TAGPEND	TAG write is already pending on this TAG
176	CC_E_NOTAG	TAG not found
178	CC_E_NOTLOCKER	Caller is not the locker of this TAG
189	CC_E_NOACCESS	TAG is READ only

Description

Use TAG_WRITE to write data to a tagged memory area. This function guarantees that the write data area is semaphored during the write operation.

C Example

```
unsigned my_fred;
unsigned status;
my_fred = 42;
.
.
.
status = TAG_WRITE (“Fred”,0,sizeof(my_fred),&my_fred,30);
```

BASIC Example

The BASIC function number is 68.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
DIM w_data            : INTEGER
.
.
.
rem * TAG_WRITE - write 4 bytes to a tag, starting at offset 0 from
rem *             w_data with a timeout of 30 seconds.
RUN AB_BAS (68,status,tag_id,0,4,ADDR(w_data),30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF(); TAG_GLOBAL_UNDEF(); TAG_READ(); TAG_LOCK(); TAG_UNLOCK(); TAG_READ_W(); TAG_WRITE_W();

TAG_WRITE_W

Writes data to a user's TAG memory area then waits for it to be read by a TAG_READ_W.

C Syntax

```
#include <copro.h>
unsigned TAG_WRITE (tag,offset,size,buffer,timeout)
    unsigned tag;
        or
    char *tag;
    unsigned offset;
    unsigned size;
    unsigned char *buffer;
    unsigned timeout;
```

Parameters

`tag`

Use to access the TAG table. This can be either the symbolic TAG or the handle returned from a TAG_LINK or TAG_DEFINE call.

`offset`

A byte offset from the start of the tagged area from where data will be written.

`size`

Specifies the number of bytes to write to the tagged area.

`buffer`

Specifies the buffer to copy the write data write to the tagged area.

`timeout`

Timeout value in seconds (valid range 0-16383). The function will timeout unless the TAG can be written before the timeout expires. The TAG may not be able to be written if another process has the TAG locked or if the corresponding TAG_READ_W from another process has not been issued. A value of CC_FOREVER (defined in COPRO.H) will cause this function to wait indefinitely until the TAG can be written.

Returns

Value	Symbolic Name	Meaning
0	CC_SUCCESS	Operation successful
120	CC_E_BADID	TAG define ID out of range
131	CC_E_TOOBIG	Data size greater than the maximum allowed
160	CC_E_INV_TO	Invalid timeout value
171	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
172	CC_E_TIME_NOREAD	Did not complete in time, the TAG not read
174	CC_E_TAGPEND	TAG write is already pending on this TAG
176	CC_E_NOTAG	TAG not found
189	CC_E_NOACCESS	TAG is READ only

Description

Use TAG_WRITE_W to write data to a tagged memory area. This function writes the data, then it waits until a corresponding TAG_READ_W is posted. This function guarantees that the write data area is semaphored during the write operation.

C Example

```
unsigned my_fred;
unsigned status;
my_fred = 42;
.
.
.
status = TAG_WRITE_W (“Fred”,0,sizeof(my_fred),&my_fred,30);
```

BASIC Example

The BASIC function code is 67.

```
DIM status            : INTEGER
DIM tag_id            : INTEGER
DIM w_data            : INTEGER
.
.
.
rem * TAG_WRITE_W - write 4 bytes to a tag, starting at offset 0 from
rem *               w_data with a timeout of 30 seconds.  This function
rem *               will return only after the tag has been read by
rem *               TAG_READ_W or a timeout.
RUN AB_BAS (67,status,tag_id,0,4,ADDR(w_data),30)
.
.
.
```

References

TAG_LINK(); TAG_DEFINE(); TAG_UNDEF(); TAG_GLOBAL_UNDEF(); TAG_READ(); TAG_LOCK(); TAG_UNLOCK(); TAG_READ_W(); TAG_WRITE();

Error Values

The following Table B.A lists all error codes (DTL, CC, and PCCC) for the control coprocessor.

Table B.A
Error Codes

DTL Errors

Decimal Value	Hex Value	Symbolic Name	Description
0	0	DTL_SUCCESS or CC_SUCCESS	Operation successful
3	3	DTL_E_DEFBAD2	Invalid number of elements to DEFINE
4	4	DTL_E_DEFBAD3	Invalid data type
5	5	DTL_E_DEFBAD4	Invalid access rights
9	9	DTL_E_DEFBADN	Invalid number of DEFINE parameters
11	B	DTL_E_FULL	DEFINE table is full
15	F	DTL_E_R_ONLY	Data item defined as READ only
16	10	DTL_E_INVTYPE	Data is invalid type for operation
17	11	DTL_E_NO_MEM	Not enough memory available
18	12	DTL_E_TIME	I/O operation did not complete in time
19	13	DTL_E_NOINIT	DEFINE table not initialized
20	14	DTL_E_BADID	Definition ID out of range
24	18	DTL_E_FAIL	I/O completed with errors
25	19	DTL_E_BADPARAM	Bad parameter value
26	1A	DTL_E_NOPARAM	Expected parameter is missing
27	1B	DTL_E_NOATMPT	I/O operation was not attempted
31	1F	DTL_E_TOOBIG	Data item greater than maximum allowed
32	20	DTL_E_NODEF	No such data item defined
38	26	DTL_E_DFBADADR	Bad DEFINE address
39	27	DTL_E_NOREINIT	DTL system already initialized
40	28	DTL_E_INPTOOLONG	DEFINE input string too long
41	29	DTL_E_CNVT	Data-conversion error
42	2A	DTL_E_GETIME	PLC-5 time invalid
50	32	DTL_E_BADDEF	Invalid use of definition

CC Errors

Decimal Value	Hex Value	Symbolic Name	Description
101	65	CC_PENDING	I/O operation in progress
117	75	CC_E_NO_MEM	Not enough memory available
118	76	CC_E_TIME	I/O operation did not complete in time
120	78	CC_E_BADID	TAG define ID out of range
124	7C	CC_E_FAIL	I/O completed with errors
127	7F	CC_E_NOATMPT	I/O operation not attempted
131	83	CC_E_TOOBIG	Data item is greater than maximum allowed
133	85	CC_E_BAD_MSGID	Message ID out of range (0-31)
141	8D	CC_E_CNVT	Data conversion error
157	9D	CC_E_NOTCONNECT	PLC is not connected or offline
159	9F	CC_E_NOEXPANDER	Expander not present
160	A0	CC_E_INV_TO	Invalid timeout value
161	A1	CC_E_INV_PORT	Invalid port address
162	A2	CC_E_INV_CTYPE	Invalid coprocessor data type
163	A3	CC_E_INV_PTYPE	Invalid PLC-5 data type
164	A4	CC_E_INV_BPI_MASK	Invalid value for BPI trigger mask
165	A5	CC_E_BAD_RACK	Rack value out of range
166	A6	CC_E_BAD_GROUP	Group value out of range
167	A7	CC_E_BAD_MODULE	Module slot value out of range
168	A8	CC_E_BAD_RETRY	Retry value out of range
171	AB	CC_E_TIME_LOCKED	Did not complete in time, TAG locked
172	AC	CC_E_TIME_NOREAD	Did not complete in time, TAG not read
173	AD	CC_E_TIME_NOWRITE	Did not complete in time, TAG not written
174	AE	CC_E_TAGPEND	TAG WRITE is already pending on this TAG
175	AF	CC_E_BADTAG	Invalid TAG name
176	B0	CC_E_NOTAG	TAG not found
177	B1	CC_E_TAGFULL	TAG table is full
178	B2	CC_E_NOTLOCKER	Caller is not the locker of this TAG
179	B3	CC_E_NOTDEFINER	Caller is not the definer of this TAG
181	B5	CC_E_MSGPEND	Message already pending
182	B6	CC_E_MSG_ABORT	Message aborted by CC_MKILL
184	B8	CC_E_DUP	Duplicate TAG
185	B9	CC_E_NOTLOCKED	Tag is not locked
186	BA	CC_E_BADACC	Bad value for TAG access
189	BD	CC_E_NOACCESS	TAG is READ only
190	BE	CC_E_SIZE	Invalid size for operation
191	BF	CC_E_TOOSMALL	Size of buffer too small
192	C0	CC_E_INVTYPE	Invalid type for operation

PCCC Errors

Decimal Value	Hex Value	Symbolic Name	Description
258	102	PCCC_E_102	Remote station did not acknowledge command
259	103	PCCC_E_103	Duplicate token holder detected on link
260	104	PCCC_E_104	Channel is disconnected from link
261	105	PCCC_E_105	Timed out waiting for a response from remote station
262	106	PCCC_E_106	Duplicate node address detected on link
263	107	PCCC_E_107	Communication channel is off-line or inactive
264	108	PCCC_E_108	Hardware fault on communication channel
272	110	PCCC_E_110	Illegal command or format, including odd address
288	120	PCCC_E_120	Host has problem and will not communicate
304	130	PCCC_E_130	Remote station host is not there, is disconnected, or shut down
320	140	PCCC_E_140	Host could not complete function due to hardware fault
336	150	PCCC_E_150	Addressing problem or memory protect rungs
352	160	PCCC_E_160	Function disallowed due to command protection selection
368	170	PCCC_E_170	Processor is in program mode
384	180	PCCC_E_180	Compatibility mode file missing or communication zone
400	190	PCCC_E_190	Remote station cannot buffer command
432	1B0	PCCC_E_1B0	Remote station problem, due to download
448	1C0	PCCC_E_1C0	Cannot execute command due to IBPs
513	201	PCCC_E_201	Illegal address format; a field has an illegal value
514	202	PCCC_E_202	Illegal address format; not enough fields specified
515	203	PCCC_E_203	Illegal address format; too many fields specified
516	204	PCCC_E_204	Illegal address; symbol not found
517	205	PCCC_E_205	Illegal address; symbol is 0 or greater than 8 characters
518	206	PCCC_E_206	Illegal address; address does not exist
519	207	PCCC_E_207	Illegal size
520	208	PCCC_E_208	Cannot complete request; situation changed
521	209	PCCC_E_209	Data is too large
522	20A	PCCC_E_20A	Size too big
523	20B	PCCC_E_20B	No access, privilege violation
524	20C	PCCC_E_20C	Resource is not available
525	20D	PCCC_E_20D	Resource is already available
526	20E	PCCC_E_20E	Command cannot be executed
527	20F	PCCC_E_20F	Overflow; histogram overflow
528	210	PCCC_E_210	No access
529	211	PCCC_E_211	Incorrect type data
530	212	PCCC_E_212	Bad parameter

Decimal Value	Hex Value	Symbolic Name	Description
531	213	PCCC_E_213	Address reference exists to deleted area
532	214	PCCC_E_214	Command execution failure for unknown reason
533	215	PCCC_E_215	Data conversion error
534	216	PCCC_E_216	1771 rack adapter not responding
535	217	PCCC_E_217	Timed out, 1771 backplane module not responding
536	218	PCCC_E_218	1771 module response was not valid: size, checksum, etc.
537	219	PCCC_E_219	Duplicated label
538	21A	PCCC_E_21A	File is open; another station owns it
539	21B	PCCC_E_21B	Another station is the program owner

BASIC Function Codes

The following Table B.B lists the BASIC function codes.

Table B.B
BASIC Function Codes

Function Name	Function Number	Description
CC_INIT	0	Initialize Control Coprocessor library
DTL_INIT	1	Initialize DTL library
DTL_C_DEFINE	2	Define a DTL data definition
DTL_UNDEF	3	Un-define a DTL data definition
DTL_DEF_AVAIL	4	Determine DTL definitions available
DTL_READ_W	5	Read data from the PLC
DTL_WRITE_W	6	Write data to the PLC
DTL_RMW_W	7	Perform read/modify/write on data
DTL_GET_WORD	8	Get a word from a byte array
DTL_GET_FLT	9	Get a floating point from a byte array
DTL_GET_3BCD	10	Convert a 3 digit BCD value to binary
DTL_GET_4BCD	11	Convert a 4 digit BCD value to binary
DTL_PUT_WORD	12	Put a word to a byte array
DTL_PUT_FLT	13	Put a floating point to a byte array
DTL_PUT_3BCD	14	Convert binary to 3 digit BCD
DTL_PUT_4BCD	15	Convert binary 4 digit BCD
DTL_SIZE	16	Get size of memory to store data item
DTL_TYPE	17	Get data type of Coprocessor data item
DTL_CLOCK	18	Set coprocessor clock to PLC clock
DTL_READ_W_IDX	20	Read any element from PLC data file
DTL_WRITE_W_IDX	21	Write any element to PLC data file
DTL_RMW_W_IDX	22	Read/modify/write any element from PLC file
BPI_DISCRETE	32	Get/Set discrete I/O word
BPI_WRITE	33	Allow Block Transfer Read
BPI_READ	34	Allow Block Transfer Write
MSG_READ_W_HANDLER	40	Initiate and process message read
MSG_READ_HANDLER	41	Initiate message read processing
MSG_WRITE_W_HANDLER	42	Initiate and process message write
MSG_WRITE_HANDLER	43	Initiate message write processing
MSG_CLR_MASK	44	Clear bit in the read/write masks
MSG_SET_MASK	45	Set bit in the read/write masks
MSG_TST_MASK	46	Test bit in the read/write masks
MSG_ZERO_MASK	47	Zero all bits in the read/write masks
MSG_WAIT	48	Wait for I/O completion of message

Function Name	Function Number	Description
TAG_DEFINE	60	Define symbolic TAG to memory
TAG_UNDEF	61	Undefine symbolic TAG of memory calling process
TAG_GLOBAL_UNDEF	62	Undefine symbolic TAG of memory any process
TAG_DEF_AVAIL	63	Determine TAG definitions available
TAG_LINK	64	Get TAG handle to symbolic TAG
TAG_READ_W	65	Read data from a symbolic TAG after a TAG_WRITE_W
TAG_READ	66	Read data from a symbolic TAG
TAG_WRITE_W	67	Write data to a symbolic TAG and wait for a TAG_READ_W
TAG_WRITE	68	Write data to a symbolic TAG
TAG_LOCK	69	Lock the TAG from concurrent access
TAG_UNLOCK	70	Unlock the locked TAG
CC_ERROR	100	Get pointer to “canned” error message
CC_ERRSTR	101	Transfer error message to user buffer
CC_DISPLAY_STR	102	Display 4 ASCII characters
CC_GET_DISPLAY_STR	103	Get the currently displayed characters
CC_DISPLAY_HEX	104	Display binary value in hexadecimal
CC_DISPLAY_EHEX	105	Display binary value in hexadecimal
CC_DISPLAY_DEC	106	Display binary value in decimal
CC_PLC_SYNC	107	Synchronize module to PLC scan
CC_PLC_STATUS	108	Get current PLC status information
CC_STATUS	111	Get current coprocessor status information
CC_EXPANDED_STATUS	112	Get current expanded coprocessor status information
CC_PLC_BTW	113	Request PLC-5 to perform block-transfer read with an I/O module
CC_PLC_BTR	114	Request PLC-5 to perform block-transfer write with an I/O module

Appendix C

Cable Connections

Appendix Objectives

This appendix provides pin assignments for ports on the main module and the serial expander module. This appendix also provides cable configurations for connecting some personal computers to the 9-pin and 25-pin ports.

Connecting to the 9-Pin COMM0 (/TERM) Port

Table C.A provides the pin assignments for the 9-pin COMM0 (/term) port on the control-coprocessor main module. Figure C.1 and Figure C.2 show cable configurations for connecting either a 9-pin or 25-pin communication port of a personal computer to the control-coprocessor main module 9-pin port.

Table C.A
Pin Assignment for 9-Pin COMM0 (/term) Port

Pin	Signal	Pin	Signal	Pin	Signal
1	DCD	4	DTR	7	RTS
2	RxD	5	Signal GND	8	CTS
3	TxD	6	DSR	9	No Connect

Cable Length Requirements

Communication for the COMM0 port complies with EIA RS-232C requirements. For all available transmission rates, you can use a cable with a maximum length up to 15 m (50 ft).

Cable Configurations

See Figure C.1 and Figure C.2 for cable configurations for the COMM0 port.

Figure C.1
Cable for a 9-Pin IBM PC/AT, T50, T60, or T47 Computer to the Control Coprocessor 9-Pin COMM0 Port

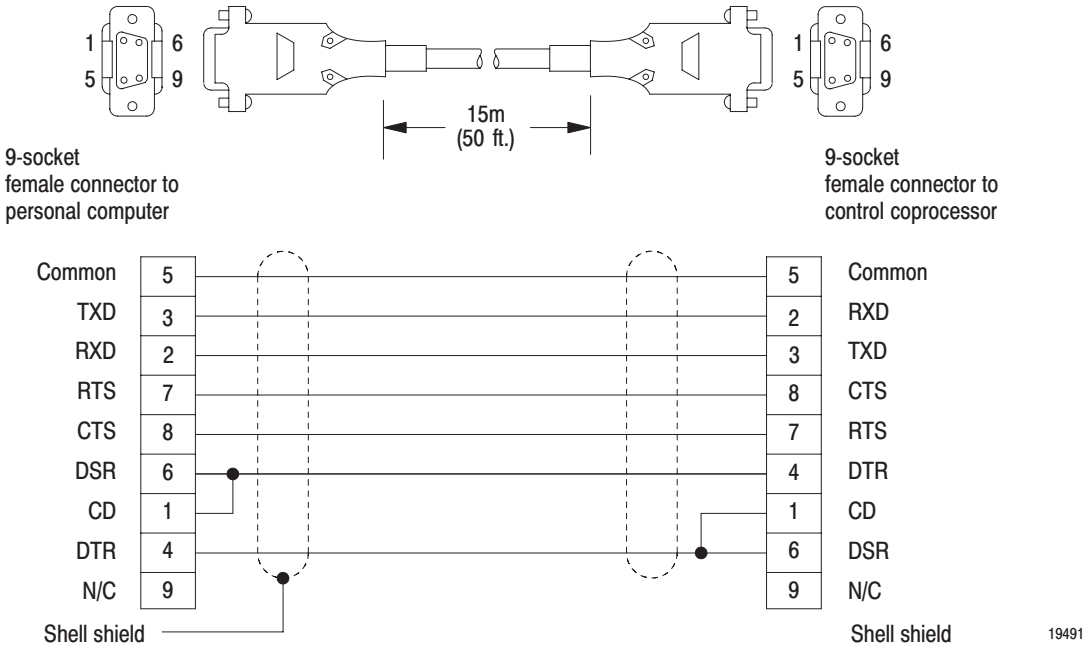
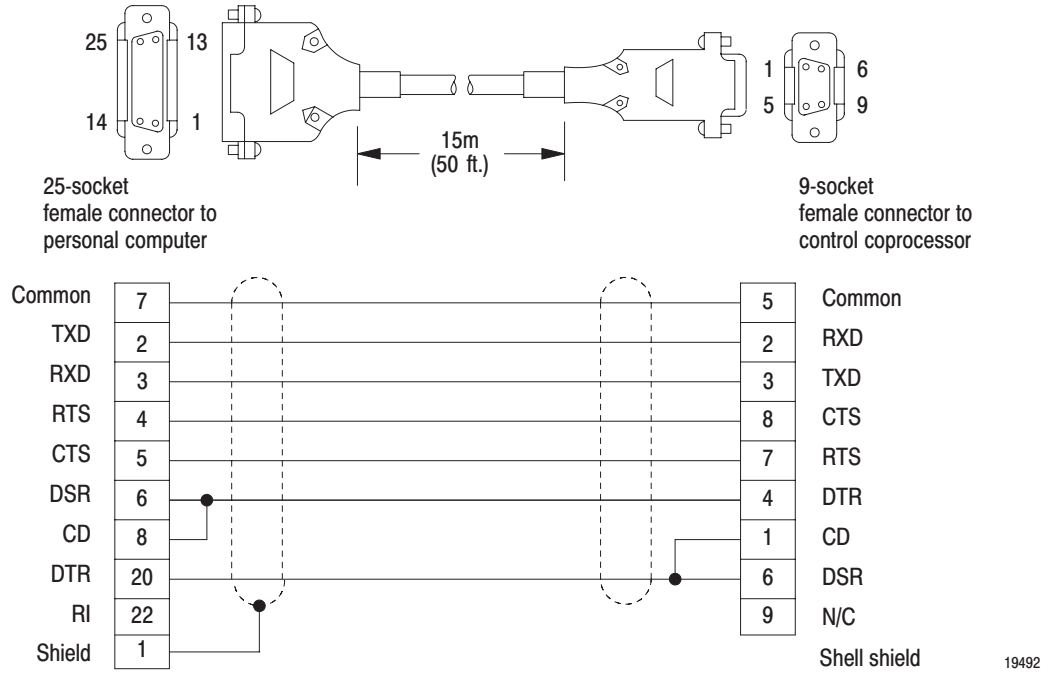


Figure C.2
Cable for a 25-Pin IBM PC/XT, VT102, VT220, 1784-T45 Computer, or Modem to the Control Coprocessor 9-Pin COMM0 Port



Connecting to the 25-Pin COMM1, 2, and 3 Ports

Table C.B provides the pin assignments for the 25-pin COMM1, COMM2, and COMM3 (/t1, /t2, /t3) ports on the main and serial expander modules.

Table C.B
Pin Assignments for 25-Pin COMM1, COMM2, and COMM3 Ports

Pin	RS-232C	RS-422A	RS-423	RS-485
1	C.GND	C.GND	C.GND	C.GND
2	TXD.OUT	TXD.OUT	TXD.OUT	RESERVED
3	RXD.IN	RXD.IN	RXD.IN	RESERVED
4	RTS.OUT	RTS.OUT	RTS.OUT	RESERVED
5	CTS.IN	CTS.IN	CTS.IN	RESERVED
6	DSR.IN	DSR.IN	DSR.IN	RESERVED
7	SIG.GND	SIG.GND	SIG.GND	SIG.GND
8	DCD.IN	DCD.IN	DCD.IN	RESERVED
9	RESERVED	RESERVED	RESERVED	RESERVED
10	NOT USED	DCD.IN′	NOT USED	NOT USED
11	RESERVED	RESERVED	RESERVED	TXRX
12	RESERVED	RESERVED	RESERVED	RESERVED
13	NOT USED	CTS.IN′	NOT USED	NOT USED
14	NOT USED	TXD.OUT′	SEND COM	NOT USED
15	RESERVED	RESERVED	RESERVED	RESERVED
16	NOT USED	RXD.IN′	REC COM	NOT USED
17	RESERVED	RESERVED	RESERVED	RESERVED
18	RESERVED	RESERVED	RESERVED	RESERVED
19	NOT USED	RTS.OUT′	NOT USED	NOT USED
20	DTR.OUT	DTR.OUT	DTR.OUT	RESERVED
21	RESERVED	RESERVED	RESERVED	RESERVED
22	NOT USED	DSR.IN′	NOT USED	NOT USED
23	NOT USED	DTR.OUT′	NOT USED	NOT USED
24	RESERVED	RESERVED	RESERVED	RESERVED
25	RESERVED	RESERVED	RESERVED	TXRX’

Cable Length Requirements

Refer to Table C.C for information on the cable lengths that you can use with the serial COMM1, COMM2, and COMM3 ports.

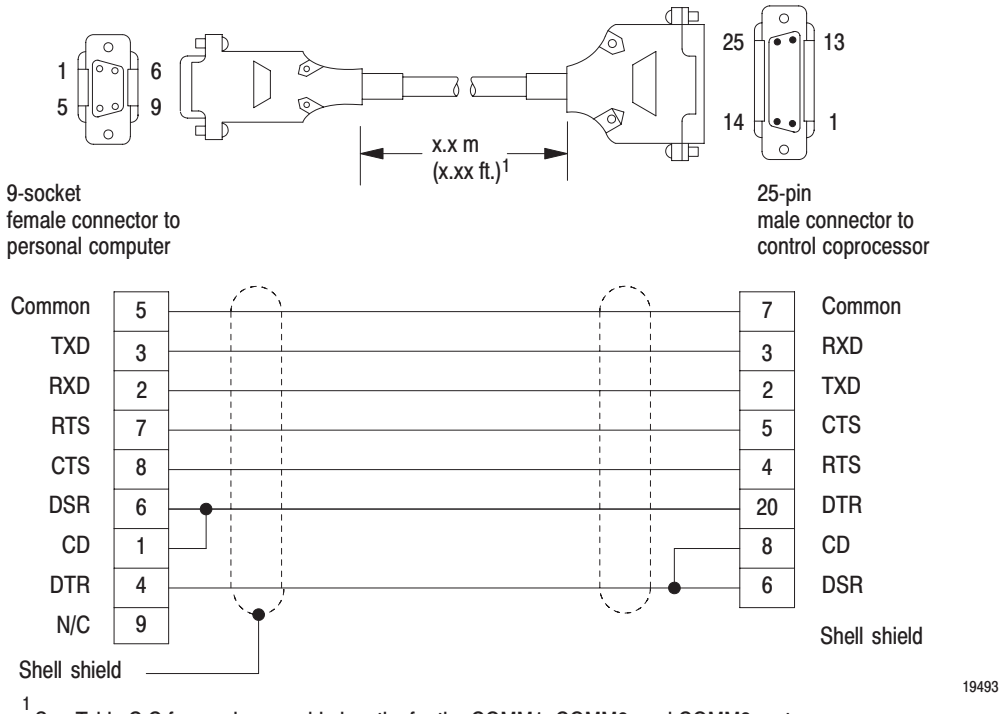
Table C.C
COMM1, COMM2, and COMM3 Maximum Cable Length

RS Communication	Transmission Rate	Maximum Cable Length
232C	all	15 m (50 ft)
422 (compatible)	19.2 kbps	61 m (200 ft)
423	9600	61 m (200 ft)
485	all	1.2 Km (4000 ft)

Cable Configurations

See Figure C.3 for an example cable configuration.

Figure C.3
Cable for a 9-Pin IBM PC/AT, T50, T60, or T47 Computer to the Control Coprocessor 25-Pin COMM1, 2, or 3 Port



[1] See Table C.C for maximum cable lengths for the COMM1, COMM2, and COMM3 ports.

Connecting to the Ethernet Port

Table C.D provides pin assignments for connecting the transceiver to the Ethernet port.

Table C.D
Pin Assignments for the Ethernet Port (Attachment Unit Interface)

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	CI-S	5	DI-A	9	CI-B	13	VP
2	CI-A	6	VC	10	DO-B	14	VS
3	DO-A	7	CO-A	11	DO-S	15	CO-B
4	DI-S	8	CO-S	12	DI-B	Shld	PG

Cable Length Requirements

Refer to Table C.E for information on the cables for the Ethernet port.

Table C.E
Ethernet Cables

Catalog Number:	Includes:
1785-TC02/A	2.0 m (6.5 ft) cable
1785-TC15/A	15.0 m (49.2 ft) cable
1785-TAS/A (kit)	Thin-wire transceiver and 2.0 m (6.5 ft) cable
1785-TAM/A (kit)	Thin-wire transceiver and 15.0 m (49.2 ft) cable
1785-TBS/A (kit)	Thick-wire transceiver and 2.0 m (6.5 ft) cable
1785-TBM/A (kit)	Thick-wire transceiver and 15.0 m (49.2 ft) cable

Appendix D

Using the PCBridge Software

Appendix Objectives

This appendix provides additional information about using the PCBridge software. Getting started using the software is covered in Chapters 3 and 4.

About PCBridge Software

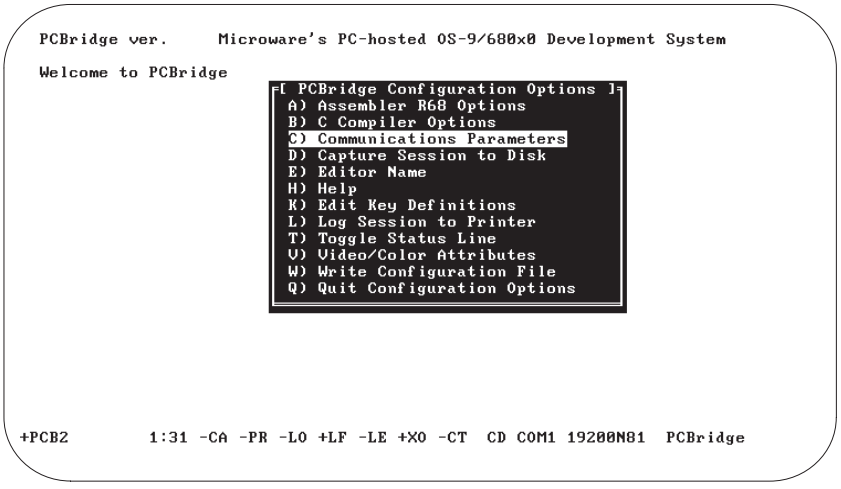
The PCBridge software is a PC-based development system for OS-9/680x0 applications. Through this software, you can access the OS-9 operating system. The PCBridge software provides a C language cross-compiler, r68 assembly and linking tools, a debugger, and a complete set of program-development utilities. These utilities include

- terminal emulation
- text and binary file transfers between PC-DOS and OS-9
- file manipulation
- session logging utilities.

Configuration Options

The following are some of the PCBridge options that you can configure. You first go to the configuration options menu. See Figure D.1.

Figure D.1
Configuration Options Menu



Edit Key Definitions

Use `k) Edit Key Definitions` to edit the function-key definition file, PCB.FNC. Select this option to invoke the chosen editor for the function key file.

Important: **`[F1]`** and **`[Alt-F2]`** through **`[Alt-F9]`** are already defined for PCBridge operations. **Do not** modify them.

You can define a string to send to the OS-9 system when any one of the following function keys is pressed:

- **`[F2]`** through **`[F12]`**
- **`[Shift-F1]`** through **`[Shift-F12]`**
- **`[Alt-F1]`** through **`[Alt-F12]`** (except **`[Alt-F2]`** through **`[Alt-F9]`**)
- **`[Ctrl-F1]`** through **`[Ctrl-F12]`**

Strings may be up to 65 characters long.

Five special characters are defined for use with the input key facility:

Use this special character:	For this purpose:
Vertical bar (|)	Represent a carriage return
Tilde (~)	Causes a one-second delay
Accent grave character (`)	Causes the PCBridge software to wait for the OS-9 system to send the next character before sending any more of the function key string
Circumflex (^)	Marks the following character as a control character
At sign (@)	Marks the following character(s) as a PCBridge command If the character following the @ is a letter, the corresponding [Alt] letter PCBridge command is executed; if the character following the @ is NOT a letter, that character is considered a delimiter and all characters in the string up to the next occurrence of the delimiter are placed in the PCBridge software's keyboard buffer

For example, the following key definition sends a **`[Ctrl-G]`** to the keyboard buffer whenever you press an **`[Alt-F1]`** key:

```
A1=^G
```

In this example, when **`[Ctrl-F1]`** is pressed, the OS-9 system waits to see a `$` prompt before executing a `dir` command.

```
C1='$@/DIR^M/
```

To use any of the special characters literally, enter a circumflex (^) and an accent grave (‘) character prior to the special character:

```
^‘
```

Log Session to the Printer

`L) Log Session to Printer` prints the PCBridge session in the same manner as the `Capture Session to Disk` option writes the PCBridge session to a file. This option works as a toggle to turn the capture session on and off. The PCBridge software writes to the DOS device PRN:. The software tries to prevent lock ups if the printer is off-line or out of paper. You can activate printer logging and capture file logging together.

Toggle the Status Line

`T) Toggle Status Line` toggles the status line display off or on. The status line indicates the current values of communications and session-logging variables.

The format of the status line is as follows:

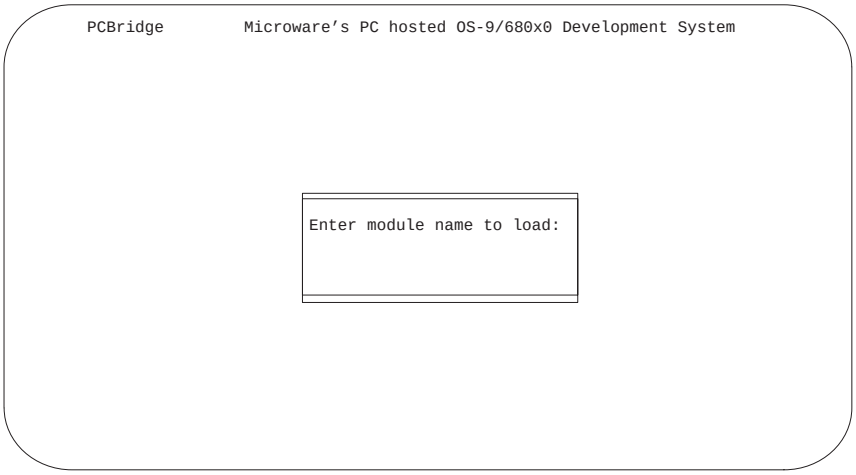
```
Scriptname HH:MM -CA -PR -LO -LF -LE -XO -CT -CD COMn baud p d b message
```

Command	Description
Scriptname	Specifies the name of a currently executing script, if any, or the name of the terminal emulation in use
HH:MM	Specifies the time of day
+CA -CA	A capture file is open No capture file is open
+PR -PR	A printer file is open No printer file is open
+LO -LO	A logging file is open No logging file is open
+LF -LF	Line feeds are added to incoming carriage returns No line feeds are added to incoming carriage returns
+LE -LE	Local echo is on Remote echo is on
+XO -XO	X-ON/X-OFF flow control is used X-ON/X-OFF flow control is turned off
+CT -CT	CTS checking is on CTS checking is off
+CD -CD	Carrier line is high Carrier dropped
COMn	Indicates the serial port currently in use
baud p d b	The baud rate, parity, data bits, and stop bits, respectively
message	Messages are displayed here to indicate unusual problems in communication

Loading Memory Module

The `L) Load Memory Module` option loads an OS-9 memory module produced by the cross-compiler or cross-assembler. If you select this option, the prompt is displayed as shown in Figure D.2.

Figure D.2
Load Memory Module



At the prompt, enter the module name. This module is transferred to the OS-9 system and loaded into memory. This does not transfer the actual module file to the disk used by the OS-9 system; it loads the module directly into OS-9 memory.

Log On Remotely to OS-9 Terminal

Use the `O) OS-9 Terminal` option to remotely log on to the OS-9 system. The PCBridge software emulates a DEC VT100 terminal. To access the PCBridge main menu, press **`[F1]`**.

Sending and Receiving Files—Transfer Tags
The PCBridge software supports a method of batching file transfers to/from the target OS-9 system. This is based on a user file called the `transfer list`. The transfer list contains text that tells the PCBridge software how to do a wide variety of file-transfer operations between the PC and OS-9. Each directory from which the PCBridge software is invoked can have its own unique transfer list. The transfer-list file, TRANSFER.LST, is in the working directory.

The PCBridge software gives you the option of transferring a single file or using a transfer tag to transfer an associated set of files. If you enter a transfer tag, the software transfers all of the files associated with that tag. The PCBridge software supports wildcards for file transfers. If a wildcard symbol is used anywhere in a file name, all of the corresponding files are transferred.

Important: A wildcard specified on a receive is processed by Kermit on the OS-9 target. A wildcard specified on a send is processed by Kermit on the PC.

A transfer list consists of:

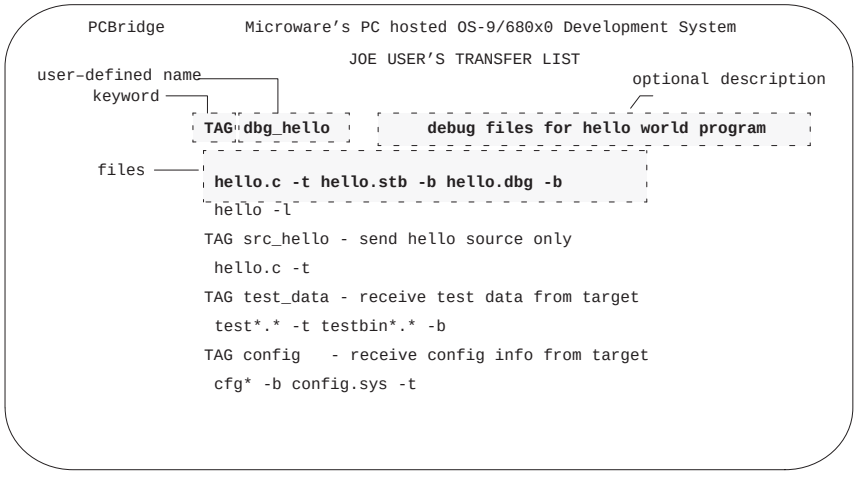
- A tag keyword (TAG), which must start in column one.
- A user-defined tag name and an optional description of the tag.
- An associated set of filename and transfer types.

You *must* specify a file type after the file name. Valid options are:

File Type	Description
-t	text file
-b	binary file
-l	send file to OS-9 and load it into memory as an OS-9 module

For example, a transfer list (TRANSFER.LST) might look like Figure D.3.

Figure D.3
Transfer List



To create or modify a transfer list, select `T) Modify Transfer List` from the main PCBridge menu. This invokes the editor on the file TRANSFER.LST in the working directory.

Modify Transfer List

Use the `T) Modify Transfer List` option to create, edit, or view the current transfer list. See page D-4 for the "Sending and Receiving Files—Transfer Tags" section. If you select this option, the PCBridge software invokes your editor on the file TRANSFER.LST in the current working directory. A transfer list entry consists of:

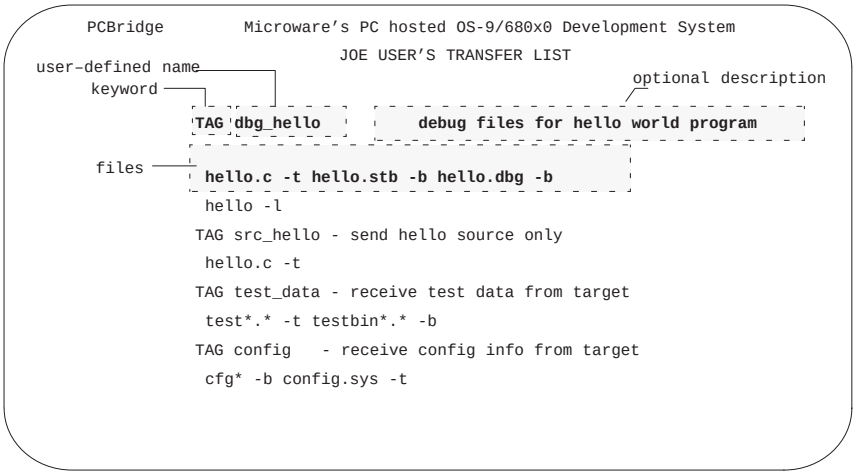
- a tag keyword (TAG) which must start in column one
- a user-defined tag name and an optional description of the tag
- an associated set of filename and transfer types

Separate entries with one or more spaces. You *must* specify a file type after the file name. Valid options are:

File Type	Description
-t	text file
-b	binary file
-l	send file to OS-9 and load it into memory as an OS-9 module

For example, a transfer list (TRANSFER.LST) might look like Figure D.4.

Figure D.4
Modify Transfer List



Modify Build List

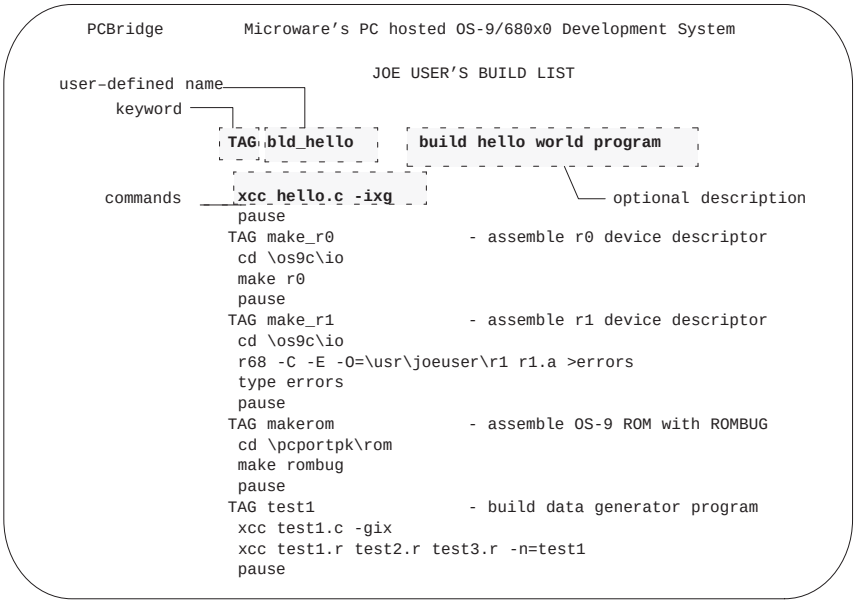
Use the `U) Modify Build List` option to create, edit, or view the current build list. Select this option to edit the file BUILD.LST in the current working directory.

A build list consists of:

- a tag keyword (TAG) which must start in column one
- a user-defined tag name and an optional description of the tag
- an associated set of commands to compile/assemble the files

For example, a build list (BUILD.LST) might look like Figure D.5.

Figure D.5
Build List



Using the Debugger

This example describes a PCBridge session in which the C program HELLO.C and its symbol file HELLO.STB are transferred to an OS-9 system and the debugger is invoked.

To use the debugger on an OS-9 module, you must compile the module with the `-g` option. This creates the symbol file necessary for debugging. The symbol file is created in the same directory as the compiled program module. By default, the C compiler command line specified in the PCBCC.BAT file does not use the `-g` option. Use the `C) Configuration Options` and `B) C Compiler Options` to add **`-g`** to the command line. It should be the same as the following line:

```
xcc -ix %1 -g
```

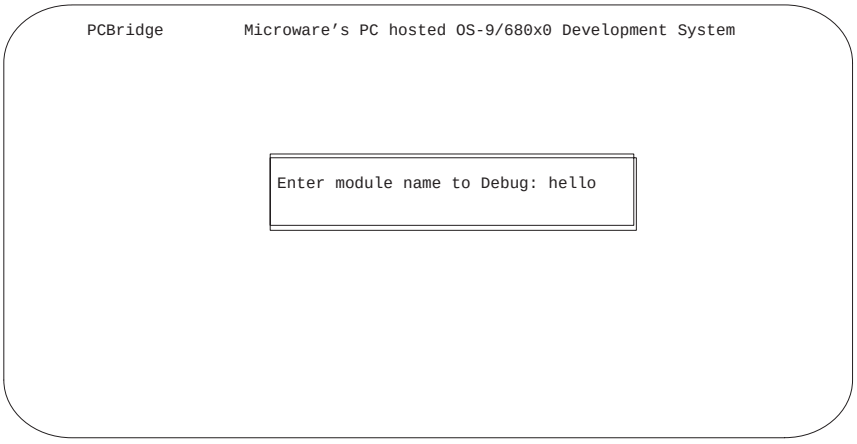
Next, use the steps provided in the previous example to compile the file HELLO.C.

You must be in OS-9 Terminal mode to use the debug utility through the PCBridge software. You must also be logged onto the OS-9 system with the same user ID as your GRPUSER DOS environment variable.

Once you have logged in and changed to the desired directory:

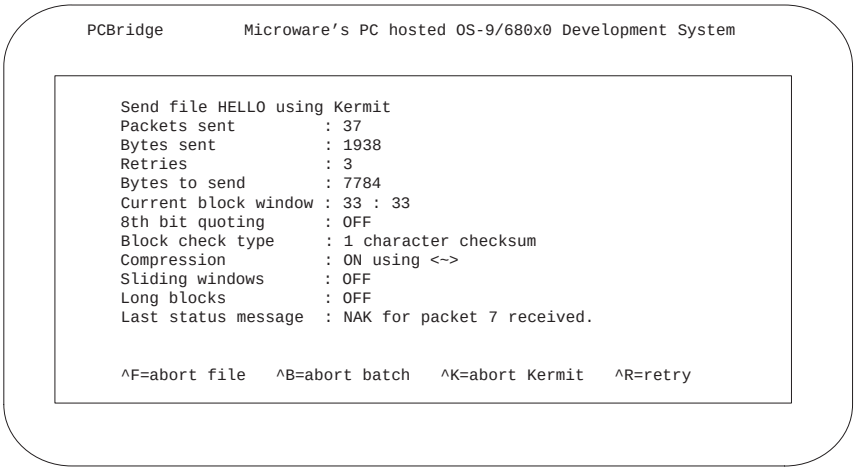
1. Press **[F1]** to get the PCBridge main menu.
2. Select the debugger `C) Debug`.
3. You are prompted for the name of the program module to debug. Enter **`hello`**.
4. You are prompted for a transfer tag associated with the debug session. Press **[Return]**. See Figure D.6.

Figure D.6
Debug Session



The PCBridge software uses Kermit to transfer the module as shown in Figure D.7.

Figure D.7
Transfer the Module



After HELLO is transferred, the PCBridge software looks for the symbol module created by the cross C compiler. If found, the symbol file HELLO.STB is transferred and the debugger is invoked. The debug prompt lets you know that the debugger has been started:

```
dgb:
```

For complete information about the debugger, see the OS-9 C Language User Manual, publication 1771-6.5.104—for example, chapter 6 of this manual explains how and where to load the online-help file for the source debugger, SRCDBG.HLP, on the control coprocessor.

Compiler Options

The PCBridge menu and communications facilities remain in memory when you compile and assemble programs, which may result in memory limitations. To overcome such memory limitations, quit the PCBridge software and call the compiler or assembler directly from the PCDOS command line. With the PCBridge software no longer in memory, there is more memory available for the task. You can call the linker directly on the command line to link several compiled modules into one OS-9 memory module.

The cross-compiler contains several options that alleviate DOS constraints. It is linked with the Phar Lap 286-DOS Extender to allow the use of extended memory.

Information on the C compiler is found in the OS-9 C User Manual, publication 1771-6.5.104. Information on the assembler and linker is found in the OS-9 Assembler/Linker User Manual, publication 1771-6.5.106.

XCC

The following options are in the cross-compiler executive (XCC):

Option	Description
-fo<string>	Fork the specified string. This option allows you to call the assembler or linker directly. For example: xcc -qfo“r68 -q file.a -o=RELS/file.r”
-lo=<opts>	Options to pass through to the linker. You can use this option to get around the 128-character DOS command-line limit. Options to pass through to the linker. You can use this option to get around the 128-character DOS command-line limit. For example, rather than typing the names of all files to link, create a file containing the file names and use a command like the following: xcc -q -lo=“-z=fnames” file.c
-po=<opts>	Options to pass through to the pre-processor. You can use this option to get around the 128 character DOS command line limit. For example, if you have several -v or -d options for the pre-processor, you can easily exceed 128 characters. Create a file containing the -v and -d lines and use a command like the following: xcc -q -po=“-z=fnames” file.c

CPP

The following option is in the macro preprocessor (CPP):

Option	Description
-z=<path>	Read file names and options from a file. For use with the -po option on XCC. For example: xcc -q -po=“-z=fnames” file.c

L68

The following option is in the OS-9 linker (L68):

Option	Description
-z=<path>	Read file names and options from a file. For use with the -lo option on XCC. For example: xcc -q -lo=“-z=fnames” file.c

Troubleshooting PCBridge Problems

Use the following table to identify PCBridge problems and apply the solution.

Problem	Solution
The PCBridge software hangs during initialization; or the PCBridge software starts, but then the screen fills with multi-colored junk or menus do not disappear when they should.	There is not enough memory to execute the PCBridge software. Generally, the PCBridge software reports that it may not have enough memory to execute before it fails.
The PCBridge software appears to execute correctly, but nothing seems to be sent out on the serial port.	Make sure that the serial cable is plugged into a serial port on your programming terminal and into the OS-9 system. Ensure that the serial port is properly installed. If the serial port is located on an add-in board, you may need to set switches or jumpers to activate the serial port correctly and to prevent conflicts with other serial port(s) that may be present in your system.
File transfers never begin.	Abort the transfer on the PCBridge software's end by pressing **`[Ctrl-F]`**. Wait a few seconds. If that does not seem to stop the transfer, try pressing **`[Ctrl-K]`**. Check the communications parameters.
File transfer aborts.	Ensure that the file type is specified correctly. It is common for a PC-DOS file to contain an extended ASCII character (one whose numeric value is greater than 127). Such characters are commonly used for drawing lines or boxes. While these characters are considered legal text, they are not legal in OS-9. Ensure that you have enough disk space on the PC to receive a file and enough OS-9 disk space when sending.
Many characters dropped in screen display.	You may find that some characters are correctly displayed, but many are simply dropped from the display and do not appear at all. Possible reasons include: • You are running a program which conflicts with the PCBridge software in some way. Try running the PCBridge software by itself. • You are running at too high a baud rate for your programming terminal to keep up with the remote system. Try using a lower baud rate. • The serial port or cable is damaged. Try replacing the cable, then try another serial port if there is one.

PCBridge Utilities

In addition to the main PCBridge program and the cross C compiler system, the PCBridge software includes several utilities:

Name	Description
binex	Binary to S-record converter
cudo	Convert text file EOL characters to UNIX, DOS, or OS-9
exbin	S-record to binary converter
fixmod	Modifies module CRC and parity
ident	Module identification utility
merge	Merge multiple files to a single file
names	List names to stdout
os9cmp	File comparison utility
os9dump	File dump utility

These utilities are not directly available from the PCBridge menu; you can invoke them from the PC-DOS command line.

binex/exbin

Convert Binary Files to S-Record File/S-Record to Binary

Syntax

```
binex [<opts>] [<inpath>] [<outpath>]
exbin [<opts>] [<inpath>] [<outpath>]
```

Function

`binex` converts binary files to S-record files. `exbin` converts S-record files to binary.

An S-record file is a type of text file that contains records representing binary data in hexadecimal form. This Motorola-standard format is often directly accepted by commercial PROM programmers, emulators, logic analyzers, and similar devices that use the RS-232 interface. It can be useful for transmitting files over data links that can only handle character-type data. You can also use it to convert OS-9 assembler or compiler generated programs to load on non-OS-9 systems.

`binex` converts the OS-9 binary file specified by `<inpath>` to a new file with S-record format. The new file is specified by `<outpath>`. S-records have a header record to store the program name for informational purposes. Each data record has an absolute memory address. This absolute memory address is meaningless to OS-9 because OS-9 uses position-independent code.

`binex` currently generates the following S-record types:

S1 records	Use a two-byte address field.
S2 records	Use a three-byte address field.
S3 records	Use a four-byte address field.
S7 records	Terminate blocks of S3 records.
S8 records	Terminate blocks of S2 records.
S9 records	Terminate blocks of S1 records.

To specify the type of S-record file to generate, use the `-s=<num>` option. `<num>` may be 1, 2, or 3, corresponding to S1, S2, or S3.

exbin is the inverse operation. `<inpath>` is assumed to be an S-record format text file that `exbin` converts to pure binary form in a new file (<outpath>). The load addresses of each data record must describe contiguous data in ascending order. `exbin` does not generate or check for the proper OS-9 module headers or CRC check value required to actually load the binary file. You can use the `ident` utility to check the validity of the modules if they are to be loaded or run. `exbin` converts any of the S-record types mentioned above.

Using either command, if both paths are omitted, standard input and output are assumed. If the second path is omitted, standard output is assumed.

Options

-?	Display the options, function, and command syntax of binex/exbin.
-a=<num>	Specify the load address in hex. This is for binex only.
-s=<num>	Specify which type of S-record format to generate. This is for binex only. <num> may be 1, 2, or 3.

Examples

The following command line generates `prog.S1` in S1 format from the binary file PROG:

```
C>binex -s1 prog prog.S1
```

The following command line generates PROG.1 in OS-9 binary format from the S1 type file PROG.S1:

```
C>exbin prog.S1 prog.1
```

cudo

Convert Text File EOL Characters to UNIX, DOS, or OS-9

Syntax

```
cudo [<opts>] {<file name>}
```

Function

`cudo` converts text files from any format to the specified format. You may specify more than one `<file name>`.

The end-of-line (EOL) characters are listed below:

Type	EOL Character	Hex
UNIX	<LF>	0x0a
DOS	<CR><LF>	0x0d0a
OS-9	<CR>	0x0d

The resulting file overwrites the original file and retains the same file name.

Functions

Options	Description
-?	Display the options, function, and command syntax
-d	Convert files to DOS format (default on DOS)
-o	Convert files to OS-9 format (default on OS-9)
-u	Convert files to UNIX format
-c	Add a <ctrl Z> to the end of a file
-q	Quiet mode
-z	Get list of input file names from stdin
-z=<file>	Get list of input file names from <path>

Examples

```
C:\> cudo -o exec.c init.c irq.c
processing:exec.c
processing:init.c
processing:irq.c
C:\> cudo -qu exec.c init.c irq.c
```

fixmod

Fix Module CRC and Parity

Syntax

```
fixmod [<opts>] {<modname> [<opts>]}
```

Function

`fixmod` verifies and updates module parity and module CRC (Cyclic Redundancy Check). You can also use it to set the access permissions and the group.user number of the owner of the module.

Use `fixmod` to update the CRC and parity of a module every time a module is patched or modified in any way. OS-9 does not recognize a module with an incorrect CRC.

You must have write access to the file before you can use `fixmod`.

Options

Option	Description
-?	Display the options, function, and command syntax of fixmod
-ua[=]<att.rev>	Change the module's attribute/revision level
-ub	Fix the sys/rev field in BASIC packed subroutine modules
-uo[=]<grp.usr>	Set the module owners group.user number to <grp.usr>
-up=<hex perm>	Set the module access permissions to <hex perm>. You must specify <hex perm> in hexadecimal
-u	Update an invalid module CRC or parity. The -u option recalculates and updates the CRC and parity. Without the u option, fixmod only verifies the CRC and parity of the module
-z	Read the module names from standard input
-z=<file>	Read the module names from <file>

Use the `-up=<hex perm>` option to set the module access permissions. You must specify `<hex perm>` in hexadecimal. You must be the owner of the module or a super user to set the access permissions. The permission field of the module header is divided into four sections from right to left:

owner permissions
group permissions
public permissions
reserved for future use

Each of these sections is divided into four fields from right to left:

read attribute
write attribute
execute attribute
reserved for future use

The entire module access permissions field is given as a four digit hexadecimal value. For example, the command `fixmod -up=555` specifies the following module access permissions field:

```
-----e-r-e-r-e-r
```

The `-uo<grp.usr>` option allows you to set the module owner's group.user number to change the ownership of a module.

Examples

The following example checks the parity and CRC for module hello.

```
C>fixmod hello
Module: hello
Calculated parity matches header parity
Calculated CRC matches module CRC
```

This example updates CRC and parity, if necessary, and changes the module owner ID to 1.85.

```
C>fixmod -uo1.85 hello
Module: hello - Fixing header parity - Fixing module CRC
```

See Also

ident

ident

Print OS-9 Module Identification

Syntax

```
ident [<opts>] {<modname>}
```

Function

`ident` displays module header information and additional information that follows the header from OS-9 memory modules.

Type **`ident`**, followed by the module name(s) to examine. `ident` displays the following information (in this order):

module size
owner
CRC bytes (with verification)
header parity (with verification)
edition
type/language, and attributes/revision
access permission

For program modules it also includes:

execution offset
data size
stack size
initialized data offset
offset to the data reference lists

`ident` also prints the interpretation of the type/language and attribute/revision bytes at the bottom of the display.

With the exception of the access permission data, all of the above fields are self-explanatory. The access permissions are divided into four sections from right to left:

owner permissions
group permissions
public permissions
reserved for future use

Each of these sections is divided into four fields from right to left:

read attribute
write attribute
execute attribute
reserved for future use

If the attribute is turned on, the first letter of the attribute (r, w, e) is displayed.

All reserved fields are displayed as dashes unless the fields are turned on. In that case, the fields are represented with question marks. In either case, the kernel ignores these fields as they are reserved for future use.

Owner permissions allow the owner to access the module. Group permissions allow anyone with the same group number as the owner to access the module. Public permissions allow access to the module regardless of the group.user number. The following example allows the owner and the group to read and execute the module, but it bars public access:

```
Permission:   $55     ---------e-r-e-r
```

Options

Option	Description
-?	Display the options, function, and command syntax of ident
-z	Read the module names from standard input
-z=<file>	Read the module names from <file>

Examples

```
C>ident hello
Header for:      hello
Module size:     $542        #1346
Owner:           1.85
Module CRC:      $BE79D0     Good CRC
Header parity:   $345A       Good parity
Edition:         $7          #7
Ty/La At/Rev     $101        $8001
Permission:      $555        -----e-r-e-r-e-r
Exec off:        $4E         #78
Data size:       $3AA        #938
Stack size:      $C00        #3072
Init. data off:  $514        #1300
Data ref. off:   $528        #1320
Prog Mod, 68000 obj, Sharable
```

merge

Merge File to MERGE.OUT File

Syntax

```
merge [<opts>] {<path>}
```

Function

`merge` copies multiple input files specified by `<path>` to a file named MERGE.OUT. `merge` is commonly used to combine several files into a single output file.

Data is copied in the same order as the pathlists specified on the command line. `merge` does no output line editing such as automatic line feed.

Options

Option	Description
-?	Display the options, function, and command syntax of merge
-z	Read the module names from standard input
-z=<file>	Read the module names from <file>

Examples

```
$ merge compile.lis asm.lis
$ merge file1 file2 file3 file4
$ merge -z=file1
```

names

List Names to stdout or file

Syntax

```
names <names>           /* list names to stdout */
```

or

```
names <names> [><fname>] /* redirect to a file */
```

Function

If <fname> is omitted, names lists the names specified on the command line to stdout. Otherwise, names redirects the output to the file specified by <fname>. You can use this command to create a file for use by the compiler/assembler/linker.

Options

None

Examples

The following example creates a file called CPPFILE and uses it during the pre-processor phase of the compiler.

```
C:\> names -v=\OSK\DEFS -v=\C600\DEFS -v=\USR\DEFS >cppfile

C:\> xcc -q -po="-z=cppfile" -r=RELS file.c
```

CPPFILE contains the following lines:

```
-v=\OSK\DEFS
-v=\C600\DEFS
-v=\USR\DEFS
```

os9cmp

Compare Binary Files

Syntax

```
os9cmp [<opts>] <path1> <path2>
```

Function

`os9cmp` opens two files and performs a comparison of the binary values of the corresponding data bytes of the files. If any differences are encountered, the file offset (address), the hexadecimal value, and the ASCII character for each byte are displayed.

The comparison ends when an end-of-file is encountered on either file. A summary of the number of bytes compared and the number of differences found is displayed.

Options

Option	Description
-?	Display the options, function, and command syntax of os9cmp
-b=<num>[k]	Assign <num>k of memory for os9cmp use. os9cmp uses a 4K memory size by default
-s	Silent mode; stop the comparison when the first mismatch occurs and print an error message

Examples

The following command uses an 8K buffer to compare FILE1 with FILE2.

```
C>os9cmp file1 file2 -b=8k

Differences
          (hex)  (ascii)
byte      #1  #2  #1  #2
========  ==  ==  ==  ==
00000019  72  6e   r   n
0000001a  73  61   s   a
0000001b  74  6c   t   l

Bytes compared:  0000002f
Bytes different: 00000003
file1 is longer
```

The following command line compares FILE1 with itself.

```
C>os9cmp file1 file1
Bytes compared:  0000002f
Bytes different: 00000000
```

os9dump

Formatted File Data Dump in Hexadecimal and ASCII

Syntax

```
os9dump [<opts>] [<path>] [<addr>]
```

Function

`os9dump` produces a formatted display of the physical data contents of `<path>`, which may be a mass storage file or any other I/O device. The `os9dump` utility is commonly used to examine the contents of non-text files.

To use this utility, type `os9dump`, the pathlist, and the address within the file if desired, of the file to display. If you omit `<path>`, standard input is used. The output is written to standard output. When you specify `<addr>`, the contents of the file display, starting with the appropriate address. `os9dump` assumes that `<addr>` is a hexadecimal number.

The data is displayed 16 bytes per line in both hexadecimal and ASCII character format. Data bytes that have non-displayable values are represented by periods in the character area.

The addresses displayed on the `os9dump` are relative to the beginning of the file. Because memory modules are position-independent and stored in files exactly as they exist in memory, the addresses shown on the dump are relative to the load addresses of the memory modules.

Options

Option	Description
-?	Display the options, function, and command syntax of os9dump
-c	Do not compress duplicate lines

Examples

The following is sample output from the command:

```
C>os9dump hello.c
 (starting  (data bytes in hexadecimal format)    (data bytes in
address)                                          ASCII format)
Addr   0 1  2 3  4 5  6 7  8 9  A B  C D  E F 0 2 4 6 8 A C E
----  ---- ---- ---- ---- ---- ---- ---- ---- ----------------
0000  2369 6e63 6c75 6465 203c 7374 6469 6f2e #include <stdio.
0010  683e 0d0a 6d61 696e 2829 0d0a 7b0d 0a09 h>..main()..{...
0020  7072 696e 7466 2822 6869 2c20 4d6f 6d2e printf("hi, Mom.
0030  2229 3b0d 0a7d 0d0a                    ");..}..
```

Index

Numbers

25-pin COMM1, 2, 3 (/t1, /t2, t3) ports, cable connections, C-3
9-pin COMM0 (/TERM) port, cable connections, C-1

A

Access Unit Interface cable. *See* AUI cable
accessing
  OS-9, command line interface, 3-7
  PCBridge, from DOS command line, 3-5
  RAM disk, program example, 4-6
  serial port, ASCII, 7-5
addresses, Ethernet port
  hardware Ethernet, 6-3, 6-11
  Internet Protocol, 6-3
API functions
  See also Application Program Interface
  defined, 5-2
    BPI, 5-2, B-1
    CC, 5-2, B-1
    DTL, 5-2, B-1
    MSG, 5-2, B-1
    TAG, 5-2, B-1
  library of functions, B-1
  when to use, 5-2
Application Program Interface
  See also API functions
  defined, 5-1
    BPI, 5-1
    CC, 5-2
    DTL, 5-1
    MSG, 5-1
    TAG, 5-2
  when to use, 5-2
applications, control coprocessor, 1-2
applying power, control coprocessor, 2-11
ASCII
  display, interpreting faults, serial expander module, 8-1
  peripheral devices, 7-5
  terminal, user interface, 1-6
AUI cable, 6-2
  See also Access Unit Interface cable

B

backplane interface
  See also BPI functions
  functions
    block transfer, 1-5, 5-6
    discrete I/O, 1-5
    how to use, 5-6
    update discrete data, 5-6
BASIC function codes, B-141
battery
  backup, main module, 1-3
  disposing, 2-4
  installing, 2-3
  replacing, 2-3
binary file, sending to control coprocessor, 4-3
block transfer, 5-6
  BPI_READ, 5-6
  BPI_WRITE, 5-6
  direct-connect mode, 1-5
  standalone mode, 1-5
BPI functions
  See also backplane interface
  block transfer, 1-5, 5-6
  discrete I/O, 1-5
  how to use, 5-6
  update discrete data, 5-6
BPI_DISCRETE, 5-6, B-3
BPI_READ, 5-6, B-5
BPI_WRITE, 5-6, B-8

C

C return values, B-137
C test program
  compiling, 4-2
  creating, 4-1
cables, C-1
  configurations
    25-pin COMM1, 2, 3 (/t1, /t2, /t3) ports, C-4
    9-pin COMM0 (/TERM) port, C-1
  connections
    25-pin COMM1, 2, 3 (/t1, /t2, /t3) ports, C-3
    9-pin COMM0 (/TERM) port, C-1
    Ethernet port, C-5
  Ethernet, 6-2
  length
    25-pin COMM1, 2, 3 (/t1, /t2, /t3) ports, C-4
    9-pin COMM0 (/TERM) port, C-1
    Ethernet port, C-5
catalog numbers, control coprocessor, 1-2
CC utility functions, 5-2
  CC_STATUS, 3-19
  clear messages, 5-13
  control coprocessor ASCII display, 5-12
  control coprocessor error, 5-12
  how to use, 5-12
  initialize control coprocessor, 5-12
  status, 5-13
  synchronization, 5-13
CC_DISPLAY _HEX, 5-12
CC_DISPLAY_DEC, 5-12, B-11
CC_DISPLAY_EHEX, 5-12, B-13
CC_DISPLAY_HEX, B-15
CC_DISPLAY_STR, 5-12, B-17
CC_ERROR, 5-12, B-19
CC_ERRSTR, 5-12, B-21
CC_EXPANDED_STATUS, 5-13, B-23
CC_GET_DISPLAY_STR, 5-12, B-25
CC_INIT, 5-12, B-27
CC_MKILL, 5-13
CC_PLC_BTR, B-28
CC_PLC_BTW, B-31
CC_PLC_STATUS, 5-13, B-34
CC_PLC_SYNC, 5-13, B-36
CC_STATUS, 3-19, 5-13, B-38
client/server applications, 6-19
  analogy, 6-20
COMM0 port, main module, 1-3
COMM1 port, main module, 1-3
  setting switches, 2-6
COMM2 and COMM3 ports, serial expander module, 1-4
  setting switches, 2-6
communication
  direct-connect mode, 1-4
  Ethernet, defined rate of, 6-1
  parameters
    configuring, 3-6
    serial ports, setting up, 7-3
    setting up, 3-6
  standalone mode, 1-4
compiling, C test program, 4-2
configuration files
  HOSTS file, 6-4
  HOSTS.EQUIV file, 6-5
  NETWORKS file, 6-6
  PROTOCOLS file, 6-6
  SERVICES file, 6-7
  STARTINET file, 6-10
configuration functions, DTL, 5-3
configuring
  communication parameters, 3-6
  control coprocessor, 3-9
  default startup parameters, 3-10
  Ethernet port, 6-12

configuring (continued)
system memory
module memory, 3-15
non-volatile, 3-11
RAM disk, 3-12
user memory, 3-13
confirming, file passage, 4-5
connecting, control processor, terminal/personal computer, 3-1
control coprocessor
applications, 1-2
applying power, 2-11
ASCII display functions, CC utility, 5-12
catalog numbers, 1-2
configuring, 3-9
default startup parameters, 3-10
system memory, 3-11
connecting, terminal/personal computer, 3-1
CSA certification, A-3
direct-connect mode, 1-4
error functions, CC utility, 5-12
hardware overview, 1-3
installing, 2-1
direct-connect mode, 2-7
standalone mode, 2-10
main module, 1-2
memory functions, DTL, 5-5
MSG instructions, 5-8
operating system, 1-7
product compatibility, A-2
product overview, 1-1
product specifications, A-1
program development software, 1-7
programming languages, 1-8
removing
direct-connect mode, 2-11
main module in standalone mode, 2-11
serial expander module, 2-11
serial expander module, 1-2
serial ports, 7-1
standalone mode, 1-4
UL certification, A-4
user interface, 1-6
conversion functions, DTL, 5-4
creating
C test program, 4-1
test directory, command line interface, 3-9
text file, 3-20
user startup file, 3-19
CSA certification, A-3

D

default startup parameters, configuring, 3-10
device names, referencing serial ports, 7-4
direct-connect mode
backplane interface
block transfer, 1-5
discrete I/O, 1-5
installing, control coprocessor, 2-7
preparing programs, 5-14
linking API functions to programs, 5-15
sample BASIC program, 5-17
sample C program, 5-15
removing, control coprocessor, 2-11
discrete I/O
direct-connect mode, 1-5
standalone mode, 1-5
disposing, battery, 2-4
DOS-based personal computer, user interface, 1-6
DTL functions, 5-2
configuration, 5-3
control coprocessor memory, 5-5
conversion, 5-4
how to use, 5-3
read/write access, 5-3
utility, 5-5
DTL_C_DEFINE, 5-3, B-40
DTL_CLOCK, 5-5, B-43

DTL_DEF_AVAIL, 5-3, B-45
DTL_GET_3BCD, 5-5, B-51
DTL_GET_4BCD, 5-5, B-53
DTL_GET_FLT, 5-5, B-47
DTL_GET_WORD, 5-5, B-49
DTL_INIT, 5-3, B-55
DTL_PUT_3BCD, 5-5, B-61
DTL_PUT_4BCD, 5-5, B-63
DTL_PUT_FLT, 5-5, B-57
DTL_PUT_WORD, 5-5, B-59
DTL_READ_W, 5-4, B-65
DTL_READ_W_IDX, B-67
DTL_RMW_W, 5-4, B-70
DTL_RMW_W_IDX, B-73
DTL_SIZE, 5-5, B-76
DTL_TYPE, 5-5, B-78
DTL_UNDEF, 5-3, B-80
DTL_WRITE_W, 5-4, B-82
DTL_WRITE_W_IDX, B-85

E

electrostatic discharge, preventing, 2-2
ESC key, 3-5
Ethernet
  cables, 6-2
    AUI, 6-2
  communication
    defined, 6-1
    defined rate of, 6-1
  configuration files, 6-4
    HOSTS file, 6-4
    HOSTS.EQUIV file, 6-5
    NETWORKS file, 6-6
    PROTOCOLS file, 6-6
    SERVICES file, 6-7
    STARTINET file, 6-10
  connecting to network
    thick-wire, 6-2
    thin-wire, 6-2
  hardware Ethernet, 6-11
  INTERD daemon, using, 6-22
  Internet
    FTP utility, 6-12
    Telnet utility, 6-17
  local area network, 6-1
  SNMPD daemon, using, 6-27
  socket library, programming, 6-19
  transceivers, 6-2
Ethernet port
  cable connections, C-5
  configuring, 6-12
  hardware Ethernet, addresses, 6-3
  Internet Protocol, addresses, 6-3
  main module, 1-3

F

fault display, serial expander module, 1-4
fault relay
  serial expander module, 1-4
  wiring, 2-10
file passage, confirming, 4-5
FTP utility
  get session, 6-16
  send session, 6-13

G

get session, FTP utility, 6-16

H

hardware Ethernet
  addresses, 6-11
  Ethernet port, addresses, 6-3
hardware overview
  control coprocessor, 1-3
  main module, 1-3
  serial expander module, 1-4
help, OS-9 utilities, command line interface, 3-7
HOSTS file, configuration files, Ethernet, 6-4
HOSTS.EQUIV file, configuration files, Ethernet, 6-5

I

initialize control coprocessor functions, CC utility, 5-12
installing
  battery, 2-3
  control coprocessor, 2-1
    direct-connect mode, 2-7
    standalone mode, 2-10
  keying bands
    main module, 2-5
    serial expander module, 2-5
  serial expander module, 2-9
  software, personal computer, 3-2
INTERCHANGE software, using, 6-22
Internet
  FTP utility, 6-12
    defined, 6-12
    get session, 6-16
    send session, 6-13
  Telnet utility, 6-17
Internet Protocol, Ethernet port, addresses, 6-3
interpreting faults
  main module, LEDs, 8-2
  serial expander module
    ASCII display, 8-1
    LEDs, 8-2

K

keying bands, installing
  main module, 2-5
  serial expander module, 2-5
keyswitch, serial expander module, 1-4

L

LEDs
  main module, 1-3
  serial expander module, 1-4
  status
    main module, 8-2
    serial expander module, 8-2
linking API functions to programs
  direct-connect mode, 5-15
  standalone mode, 5-18
local area network, Ethernet, 6-1
local chassis, standalone mode, 1-5
lock/unlock functions, TAG, 5-11

M

main module
  battery backup, 1-3
  COMM0 port, 1-3
  COMM1 port, 1-3
  control coprocessor, 1-3
  Ethernet port, 1-3
  hardware overview, 1-3
  in standalone mode, removing, 2-11
  interpreting faults, LEDs, 8-2
  keying bands, installing, 2-5
  LEDs, 1-3
  RAM memory, 1-3
  reset switch, 1-3
  setting switches, COMM1 port, 2-6
  standalone mode, removing, 2-11
memory module, loading via PCBridge, D-4
modes of communication
  direct-connect mode, 1-4
  standalone mode, 1-4
module memory
  NVMM utility, 3-16
  system memory, configuring, 3-15
MSG instructions, 5-2
  control coprocessor MSG functions, 5-8
  how to use, 5-7
  PLC-5 programmable controller, 5-7
MSG_CLR_MASK, B-88
MSG_READ_HANDLER, 5-9, B-90
MSG_READ_W_HANDLER, 5-9, B-94
MSG_SET_MASK, B-98
MSG_TST_MASK, B-100

MSG_WAIT, B-102
MSG_WRITE_HANDLER, 5-9, B-105
MSG_WRITE_W_HANDLER, 5-9, B-109
MSG_ZERO_MASK, B-113

N

NETWORKS file, configuration files, Ethernet, 6-6
non-volatile, system memory, configuring, 3-11
NVMM utility, module memory, 3-16

O

OS-9, command line interface
  accessing, 3-7
  creating test directory, 3-9
  help, 3-7
  returning to PCBridge, 3-9
  setting time, 3-8
OS–9, terminal option, D-4

P

PCBridge
  accessing, from DOS command line, 3-5
  batch file transfers, D-4
  binary file comparison, D-21
  buildlist, D-6, D-7
  combine several files, D-19
  compare binary files, D-21
  compiler options
    cpp, D-10
    l68, D-10
    xcc, D-10
  convert
    binary file to s–record, D-12
    s–record to binary, D-12
    text files, D-14
  cross compiler options, D-9
  data dump, D-22
  display module information, D-17
  DOS constraints, D-9
  edit, key definitions, D-2
  end–of–line (EOL), characters, D-14
  F1 function key, D-4
  function key, D-2
  group permissions, D-18
  highlight menu item, 3-5
  list names to stdout or file, D-20
  load memory module, D-4
  log session to printer, D-3
  module parity and CRC, verify and update, D-15
  OS–9 remote logon, D-4
  owner permissions, D-18
  PCB.FNC, D-2
  PCBCC.BAT, D-7
  program development software, 1-7
  public permissions, D-18
  s–record files, D-12
  select menu item, 3-5
  status line, description, D-3
  strings, D-2
  transfer
    list, D-4
    tag, D-4
  transfer list
    create, D-6
    edit, D-6
    modify, D-6
  troubleshooting, D-11
  update CRC and parity, D-15
  using the debugger, D-7
  utilities
    binex, D-12
    cudo, D-14
    exbin, D-12
    fixmod, D-15
    ident, D-17
    merge, D-19
    names, D-20
    os9cmp, D-21

PCBridge (continued)
utilities (continued)
os9dump, D-22
view, transfer list, D-6
wildcards, D-4
PLC programmable controller
direct-connect mode, 1-4
backplane interface, 1-5
standalone mode, 1-4
PLC-5 programmable controller, MSG instructions, 5-7
pointers, using, B-2
power supply, 2-2
preparing programs
direct-connect mode, 5-14
linking API functions to programs, 5-15
sample BASIC program, 5-17
sample C program, 5-15
standalone mode, 5-18
linking API functions to programs, 5-18
sample BASIC program, 5-20
sample C program, 5-18
sample control logic program, 5-21
preventing, electrostatic discharge, 2-2
product
certification, UL, A-4
compatibility, A-2
overview, control coprocessor, 1-1
specifications, control coprocessor, A-1
program development software, PCBridge, 1-7
programming
environment
compiling, C test program, 4-2
creating, C test program, 4-1
sending, C test program, 4-3
languages, 1-8
overview
languages, 1-8
operating system, 1-7
program development software, 1-7
user interface, 1-6
socket library, 6-19
PROTOCOLS file, configuration files, Ethernet, 6-6

R

RAM
accessing, program example, 4-6
main module, 1-3
system memory, configuring, 3-12
read/write
access functions, DTL, 5-3
TAG, 5-11
remote chassis, standalone mode, 1-5
removing, control coprocessor
direct-connect mode, 2-11
main module in standalone mode, 2-11
serial expander module, 2-11
replacing, battery, 2-3
reset switch, main module, 1-3
return, to previous screen, 3-5
returning to PCBridge, command line interface, 3-9

S

send session, FTP utility, 6-13
sending
binary file to control coprocessor, 4-3
text file to control coprocessor, 3-21
serial expander module
COMM2 and COMM3 ports, 1-4
control coprocessor, 1-4
fault display, 1-4
fault relay, 1-4
hardware overview, 1-4
installing, 2-9
interpreting faults

serial expander module (continued)
ASCII display, 8-1
LEDs, 8-2
keying bands, installing, 2-5
keyswitch, 1-4
LEDs, 1-4
removing, control coprocessor, 2-11
setting switches
COMM2 port, 2-6
COMM3 port, 2-6
serial ports, 7-1
ASCII, 7-1
accessing a port, 7-5
example program, 7-5
using, 7-5
device names, 7-4
RS-485 communication, 7-10
example code, 7-12
RS-422 communication, 7-17
SERVICES file, configuration files, Ethernet, 6-7
setting switches
main module, COMM1 port, 2-6
serial expander module
COMM2 port, 2-6
COMM3 port, 2-6
setting time, OS-9 command line interface, 3-8
setting up
communication parameters, 3-6, 7-3
tmode, 7-3
xmode, 7-3
password file, 3-20
socket library
client/server applications, 6-19
analogy, 6-20
programming, 6-19
software
installing, personal computer, 3-2
PCBridge, accessing from DOS command line, 3-5
standalone mode
backplane interface
block transfer, 1-5
discrete I/O, 1-5
installing, 2-10
local chassis, 1-5
preparing programs, 5-18
linking API functions to programs, 5-18
sample BASIC program, 5-20
sample C program, 5-18
sample control logic program, 5-21
remote chassis, 1-5
STARTINET file, configuration files, Ethernet, 6-10
status functions
CC utility, 5-13
view coprocessor current status, 3-19
synchronization functions, CC utility, 5-13
system memory, configuring, 3-11
module memory, 3-15
RAM disk, 3-12
user memory, 3-13

T

table configuration functions, TAG, 5-11
tag, keyword, D-6
TAG functions, 5-2
how to use, 5-10
lock/unlock, 5-11
read/write, 5-11
table configuration, 5-11
TAG_DEF_AVAIL, 5-11, B-115
TAG_DEFINE, 5-11, B-116
TAG_GLOBAL_UNDEF, B-119
TAG_LINK, 5-11, B-121
TAG_LOCK, 5-11, B-123
TAG_READ, 5-11, B-125
TAG_READ_W, 5-11, B-127
TAG_UNDEF, 5-11, B-129
TAG_UNDEF_GLOBAL, 5-11

TAG_UNLOCK, 5-11, B-131
TAG_WRITE, 5-11, B-133
TAG_WRITE_W, 5-11, B-135
Telnet utility, 6-17
text file
  creating, 3-20
  sending, 3-21
tmode, communication parameters, setting up, 7-3
transceiver, Ethernet, 6-2
transfer, list, file type, D-6

U

UL certification, A-4
user interface
  ASCII terminal, 1-6
  DOS-based personal computer, 1-6
user memory, system memory, configuring, 3-13
user startup file
  creating, 3-19
  example, 3-19
  setting up password file, 3-20
using
  INTERD daemon, 6-22
  pointers, B-2
  serial ports
    ASCII, 7-5
    RS-485 communications, 7-10
    RS-422 communications, 7-17
  SNMPD daemon, 6-27
utility function
  CC_STATUS, 3-19
  DTL, 5-5

W

wiring, fault relay, 2-10

X

xmode, communication parameters, setting up, 7-3

ASCII Character Codes

Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex
[ctrl-@] NUL	0	00	SPACE	32	20	@	64	40	‘	96	60
[ctrl-A] SOH	1	01	!	33	21	A	65	41	a	97	61
[ctrl-B] STX	2	02	”	34	22	B	66	42	c	99	63
[ctrl-C] ETX	3	03	#	35	23	C	67	43	b	98	62
[ctrl-D] EOT	4	04	$	36	24	D	68	44	d	100	64
[ctrl-E] ENQ	5	05	%	37	25	E	69	45	e	101	65
[ctrl-F] ACK	6	06	&	38	26	F	70	56	f	102	66
[ctrl-G] BEL	7	07	’	39	27	G	71	47	g	103	67
[ctrl-H] BS	8	08	(	40	28	H	72	48	h	104	68
[ctrl-I] HT	9	09	)	41	29	I	73	49	i	105	69
[ctrl-J] LF	10	0A	*	42	2A	J	74	4A	j	106	6A
[ctrl-K] VT	11	0B	+	43	2B	K	75	4B	k	107	6B
[ctrl-L] FF	12	0C	,	44	2C	L	76	4C	l	108	6C
[ctrl-M] CR	13	0D	–	45	2D	M	77	4D	m	109	6D
[ctrl-N] SO	14	0E	.	46	2E	N	78	4E	n	110	6E
[ctrl-O] SI	15	0F	/	47	2F	O	79	4F	o	111	6F
[ctrl-P] DLE	16	10	0	48	30	P	80	50	p	112	70
[ctrl-Q] DC1	17	11	1	49	31	Q	81	51	q	113	71
[ctrl-R] DC2	18	12	2	50	32	R	82	52	r	114	72
[ctrl-S] DC3	19	13	3	51	33	S	83	53	s	115	73
[ctrl-T] DC4	20	14	4	52	34	T	84	54	t	116	74
[ctrl-U] NAK	21	15	5	53	35	U	85	55	u	117	75
[ctrl-V] SYN	22	16	6	54	36	V	86	56	v	118	76
[ctrl-W] ETB	23	17	7	55	37	W	87	57	w	119	77
[ctrl-X] CAN	24	18	8	56	38	X	88	58	x	120	78
[ctrl-Y] EM	25	19	9	57	39	Y	89	59	y	121	79
[ctrl-Z] SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ctrl-[ESC	27	1B	;	59	3B	[	91	5B	{	123	7B
[ctrl-\] FS	28	1C	<	60	3C	\	92	5C	|	124	7C
ctrl-] GS	29	1D	=	61	3D	]	93	5D	}	125	7D
[ctrl-^] RS	30	1E	>	62	3E	^	94	5E	~	126	7E
[ctrl-_] US	31	1F	?	63	3F	_	95	5F	DEL	127	7F

Publication 1771-6.5.95—December 1994
Supersedes 1771-6.5.95—March 1993

PN 955119-26