2.0

APPLICATION SPECIFIC FUNCTION BLOCK MANUAL

COMM900

GIDDINGS & LEWIS, INC.

NOTE

Progress is an ongoing commitment at Giddings & Lewis. We continually strive to offer the most advanced products in the industry; therefore, mformation in this document is subject to change without notice. The illustrations and specifications are not binding in detail. Giddings and Lewis shall not be liable for any technical or editorial omissions occurrıng in this document, nor for any consequential or incidental damages resulting from the use of this document.

DO NOT ATTEMPT to use any Giddings & Lewis product until the use of such product is completely understood. It is the responsibility of the user to make certain proper operation practices are understood. Giddings & Lewis products should be used only by qualified personnel and for the express purpose for which said products were designed.

Should information not covered in this document be required, contact the Customer Service Department, Giddings & Lewis, 660 South Military Road, P.O. Box 1658, Fond du Lac, WI 54936-1658. Giddings & Lewis can be reached by telephone at (920) 921-7100.

401-55377-00

2698

Table of Contents

Application Specific Function Block Guidelines

Installation

The following guidelines are recommended ways of working with Application Specific Function Blocks (ASFBs) from Giddings & Lewis.

1. Make a back up copy of the ASFB disk you receive and store the original in a safe place.
2. The disk you receive with the ASFB package will include the following:

   1. ASFBs directory(s) containing:
      - .LIB file(s) containing the ASFB(s)
      - source .LDO(s) from which the ASFB(s) was made
   2. EXAMPLES directory(s) containing:
      - example LDO(s) with the ASFB(s) incorporated into the ladder which you can then use to begin programming from or merge with an existing application ladder

   It is recommended that you copy the .LIB and the source LDO files to your hard drive on the PC in the following way. Remember that ASFB libraries (.LIB) files and source (.LDO) files must be kept in the same directory.

   - Create a directory that will hold all ASFB LIBs and source LDOs. For example, you may have the Motion' ASFB package and the Communication ASFB package. Copy the appropriate files on the disks to a directory on your PC called ASFB.

     When you installed PiCPro, the PiCLib statement was automatically entered in your autoexec.bat file as shown below:

     SET PICLIB=C:\PICLIB

     NOTE: If you chose to alter your PICLIB statement during installation, it will look different than what appears above.

     Now add the ASFB directory to your PICLIB = statement as shown below:

     SET PICLIB=C:\PICLIB;C:\ASFB

   - Put the example file(s) in your working directory. For example, if you always run PiCPro from the directory which holds all your LDO files, then copy all the ASFB example LDOs to the LDO directory.

Revisions

3. The first three networks of each ASFB source ladder provide the following information.

Network 1

The first network is used to keep a revision history of the ASFB. Revisions can be made by Giddings & Lewis personnel or by you.

The network identifies the ASFB, lists the requirements for using this ASFB, the name of the library the ASFB is stored in, and the revision history.

The revision history includes the date, ASFB version (see below), the version of PiCPro used while making the ASFB, and comments about what the revision involved.

When an ASFB is revised, the number of the first input (EN _ _or RQ_ _) to the function block is changed in the software declarations table. The range of numbers available for Giddings & Lewis personnel is 00 to 49. The range of numbers available for you is 50 to 99. See chart below.

Revision	Giddings & Lewis revisions	User revisions
1st	EN00	EN50
2nd	EN01	EN51
.	.	
.	.	
	.	
50th	EN49	EN99

```
Network 1
|...1.......
X-Name ASFB Source Revision History

Located in Library X-LIB

Requirements:
PiCPro Ver 4.0 or higher


Date        Version    Using PiCPro    Comments
----        -------    ------------    --------
MM-DD-YY     EN00          4.1         Original
```

Network 2

The second network describes what you should do if you want to make a revision to the ASFB.

```
|-----------
|...2.......
```

If you revise the ASFB, do the following:

1. Do a 'M'odule, save 'A's in order to save the original ASFB before you begin modifying.
2. Change the number on the first input to the ASFB in the software declarations table to a 50 or greater (for example, EN00 would be changed to EN50).
3. Update the revision history in network 1.

ASFB Input/Output Descriptions

Network 3

The third network describes the ASFB and defines all the inputs and outputs to the function block.

```
|-----------
|...3.......
```

ASFB Description

INPUTS:

Name	Data Type	Definition
EN00	BOOL	enables execution

OUTPUTS:

Name	Data Type	Definition
OK	BOOL	execution complete

4. When you are ready to use the ASFB in your application, there are several approaches you can take as shown below.

   - Create a new application LDO starting with the example LDO for the ASFB package. The advantage is that the software declarations table for the ASFB has been entered for you.

     NOTE: To keep the original example LDO, use the 'save As' command. This copies the example LDO to an LDO with the application name you give it.

   - If you already have an application LDO, merge the example LDO with the application LDO using the optional LDOMERGE software package. The software declaration tables for both LDOs will also merge.

   - Enter the ASFB into your application LDO.

     NOTE: This method is not recommended if the software declarations table is lengthy. It requires that you manually enter all the inputs and outputs to the ASFB in the table. With some packages, this is time-consuming. Any structure, array, array of structures, or strings must be entered *exactly* as it appears in the original table. This is critical to the correct functioning of the ASFB.

Chapter 1 - Introduction

1.1 - Overview

The COMM900 communication software package provides the software tools necessary to perform data communications with PiCs. The communication can be either serially through RS-232 ports or over a network. These two methods of setting up the communications are summarized below.

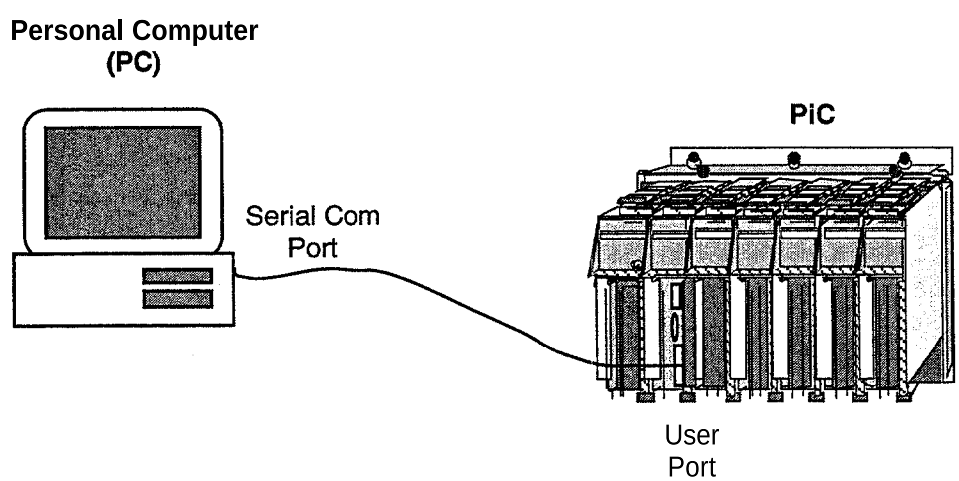
	COMM900 Package	Hardware Connection	Format
1.	Serial Software	RS-232 Port on the PC and PiC	PC to one PiC
2.	Network Software	Peer-to-Peer Network (ARCNET) Connector on the PC and PiC(s)	PC to multiple PiCs or PiC-to-PiC

Serial Communication

The serial software allows communication to occur between a host computer (PC) and one PiC as shown in Figure l-l. This is a single drop connection with only one PiC connected to a host computer at a time. Other features include:

- A single RS-232 connection from corn port 1 or 2 of the PC to the USER PORT or a Serial Communications Module port of the PiC.
- Baud rates of 1200, 2400, 4800, 9600, 19200 (use hardware handshaking at 19200).
- Wiring length limited by RS-232 (typically 50 feet at 9600 baud).

Figure l-l. RS-232 Serial Communications (Single Drop)



Network Communication

The network software can be used in two ways:

1. Multiple PiCs connected to a host computer as shown in Figure l-2.
2. Multiple PiCs connected without a host computer as shown in Figure l-3.

- The physical connection is:

    1. From the optional ARCNET connector on the PiC CPU to the ARCNET board installed in the ISA slot of the host computer
    2. Between the optional ARCNET connectors on the PiCs

This is a multi-drop connection with up to 254 PiCs connected to a host computer or 255 PiCs connected to each other at one time. Other features include:

- Typical communication rates are 2.5 Mbits/second.
- Connection options include the standard twisted pair (distances up to 400 feet), or the optional coaxial cable and fiber optics (distances up to 4 miles). Refer to the PiC900 Hardware Manual for configuration options.
- Greater noise immunity than the serial RS-232 connections.

Figure l-2. Network Connections with Host Computer

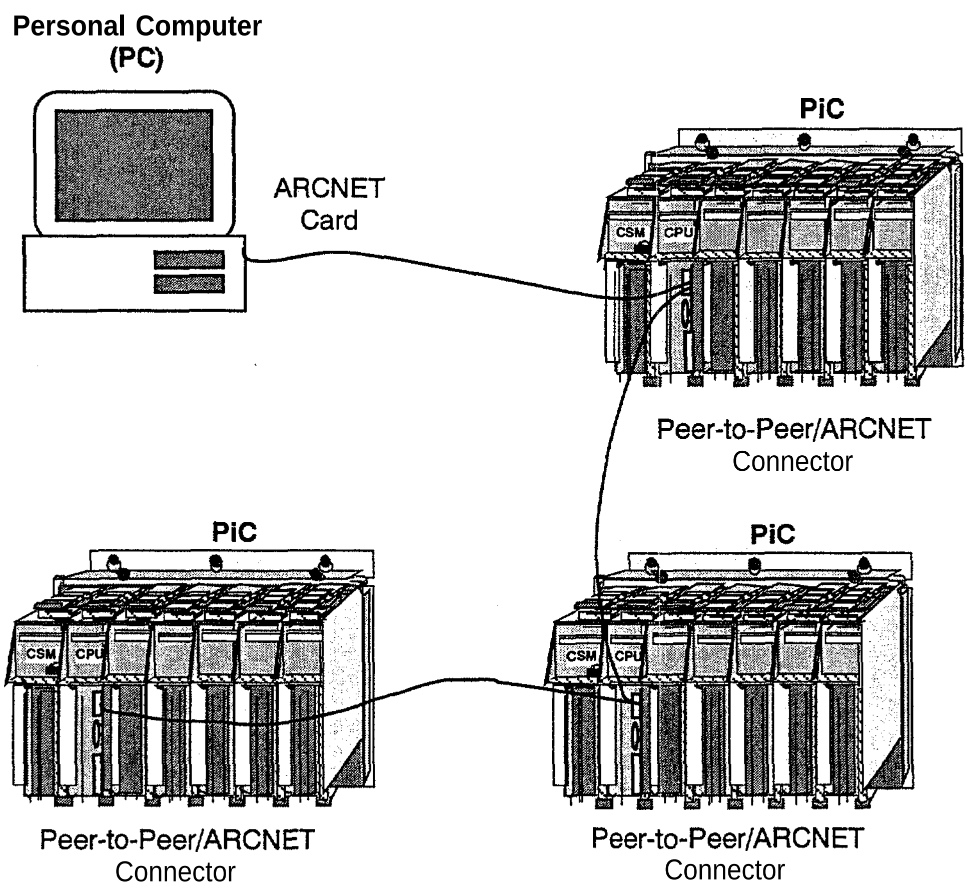
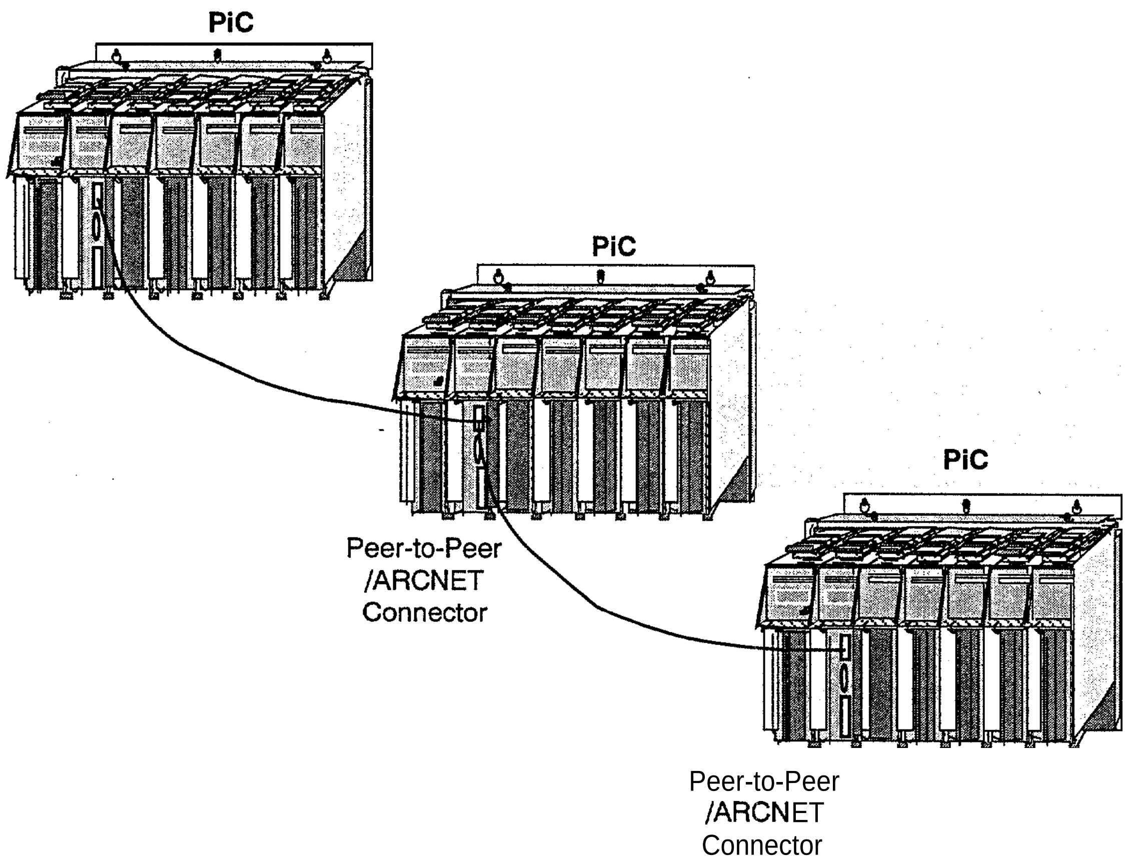


Figure 1-3. Network Connections without a Host Computer



NOTE: A single PiC may be connected via ARCNET to a PC where the features listed under network communications are required, e.g., wiring distance or speed.

1.2 - Software interface

Standard Components

The PC

The PC or host computer is configured to communicate to a PiC using software communication drivers provided as part of the COMM900 package. Communication drivers are used to implement both the ARCNET and serial communications. They allow the host computer to operate like a transmitter. It can then send READ or WRITE messages.

The COMM 900 communications driver may be run under DOS, Windows, Windows 95, or Windows NT.

The host computer must initiate all messages. No unsolicited messages are supported. If an application requires a PiC to send information to the host, the host computer would have to poll the value at a predetermined time interval.

The PiC

A PiC communicates to a host computer or another PiC via function blocks called Application Specific Function Blocks (ASFBs). They are called within the main application ladder.

There are two transceiver ASFBs:

C_NETXCV for use with network communication
C_SERXCV for use with serial communication.

A transceiver function block receives messages from external devices connected to either the ARCNET port or the serial port, executes the command , and responds to the sender with the status. A message will contain a command to be performed. A READ command is sent to a PiC when an external device needs to know the value of a certain variable(s) stored within the PiC. A WRITE command is sent to a PiC when an external device needs to change the value of a certain variable(s).

There is one transmitter ASFB:

C_NETXFR for PiC-to-PiC transfers.

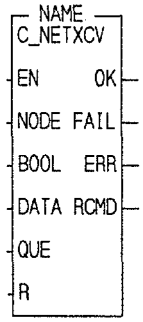
The transmitter function block sends messages from one PiC to another PiC connected via the ARCNET port. No other transport media are supported at this time. A message will contain a command to be performed. A READ command is sent to another PiC when one PiC needs to know the value of a variable stored within the other PiC. A WRITE command is sent to another PiC when the one PiC needs to change the value of a certain variable within the other PiC. A transmitter can only be used when implemented with ARCNET.

Note that any PiC used in PiC-to-PiC transfers may also communicate with and respond to messages originating from a host computer.

The COMM900 software includes the ASFBs shown below. You install the appropriate ASFB(s) and use them in your application ladder. Or you can use the example LDO which has the ASFBs incorporated into its logic.

Network Communications ASFBs

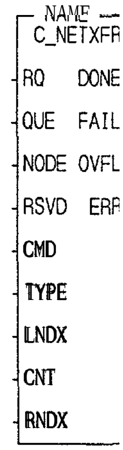
Transceiver ASFB
C NETXCV



Performs network data transfers between the PiC in which it is enabled and a PC or another PiC.

Establishes the network ID of the PiC in which it is executed.

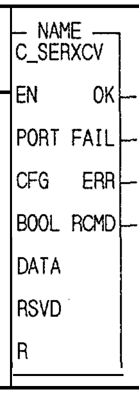
Transmitter ASFB
C_NETXFR



Allows the PiC in which it is transitioned to initiate a data transfer request to another PiC.

Serial Communications ASFB

Transceiver ASFB
C SERXCV



Communications-Serial

Performs serial data transfers between the PiC in which it is enabled and a PC.

Optional Components

The following section describes optional communications drivers available for use with the COMM900 package.

Giddings & Lewis PiC Dynamic Link Library (DLL) for Windows

This Dynamic Link Library allows a Microsoft Windows application developed using Microsoft Visual Basic or Microsoft C++ to communicate to one or more PiC(s) that incorporate the COMM900 function blocks. The 16-bit DLL is designed to run on Windows 3.1 and the 32-bit DLL is designed to run on Windows 95 or NT.

The DLL is an additional driver that replaces the MS-DOS based communication drivers that come with the standard COMM900 ASFB package as described above. The communication interface works just like the interface defined for the standard MS-DOS based drivers, but this driver allows applications to run under the Window GUI interface.

Drivers are available for both ARCNET and serial communication options. Please refer to the documentation for your application development tool kit for information on how to link your application to a standard DLL.

Giddings & Lewis PiC Dynamic Data Exchange (DDE) Server for Windows

The PiC DDE Server is a Microsoft Windows application which acts as a DDE (Dynamic Data Exchange) server and allows other Windows applications to access data on Giddings & Lewis PiCs via the COMM900 protocol. The 16-bit server is designed to run on Microsoft Windows 3.1 and the 32-bit server is designed to run on Windows 95 or NT. These servers may be used by any Microsoft Windows program which is capable of acting as a DDE Client.

The DDE server physically communicates to a Giddings & Lewis PiC through a serial port or an industry standard ARCNET port. The ARCNET configuration allows the server to communicate to a network of PiCs and a serial configuration is a single-drop connection. The ladder running in the PiC must incorporate the COMM900 ASFBs (version 1.3 or higher) package.
NOTE: Communication to a PiC that incorporates a PiCPro Network ID feature requires COMM900 ASFB version 2.0 or higher.

More information on the specific features and programming of the PiC DDE Server for Windows can be found in the PiC DDE Server User Manual.

Giddings & Lewis PiC Dynamic Data Exchange (DDE) Server for Wonderware®

The PiC DDE server for Wonderware is software compatible with the PiC DDE Server for Windows as described above. The difference is that this version has been developed to optimize the data communications between the PiC and the DDE server. Fast DDE is used to communicate blocks of data. This implementation provides better data throughput for applications performing large data collection when using the Wonderware Man Machine Interface software.

NOTE: Even though many software developers have also implemented the Fast DDE protocol, the Wonderware version of the DDE server should only be used when interfacing to the Wonderware MMI.

1.3 - Hardware requirements and installation

Requirements

Hardware requirements for COMM900 are as follows.

	Serial	Network
• **PiC**	PiC User Port (or the 2 or 4 ports on the optional serial communications module)	PiC with peer-to-peer communications capabilities
• **PC**	100% IBM compatible	100 % IBM compatible with ARCNET card
• **Cable/wire**	RS-232 25-pin or 9-pin D connector to 10-pin CPU or 40-pin Serial Communications Module screw terminal connector NOTE: Hardware handshaking is required for baud rates of 19200.	See the descriptions at Network

Connections

Serial Communications Connections

For communication rates up to 9600 baud, an ordinary three wire serial cable is used to connect the PiC to the PC. Pinouts for the 9-pin or 25-pin PC connector to the 10-pin screw terminal connector are shown below. If using hardware handshaking, connect the PiC RTS to the PC CTS and the PiC CTS to the PC RTS.

NOTE: The PiC serial User Port can be used with baud rates up to 19.2K. Above 19.2K baud, a Serial Communications Module must be installed in the main rack. All communication rates above 19.2K baud require hardware handshaking.

- **Cables**

On the PiC Screw terminal connector	to a 9-pin	OR	to a 25-pin
GND <-------------------------------->	pin 5	GND	pin 7
Receive data <-------------------->	pin 3	Transmit data	pin 2
Transmit data <--- ---------- ------>	pin 2	Receive data	pin 3

Network Communications Connections

Communications capabilities on the PiC CPU Module

The PiC CPU module must have ARCNET capabilities. Refer to the PiC900 Hardware Manual for wiring ARCNET.

ARCNET card for the PC

Refer to the PiC900 Hardware Manual for recommended ARCNET board suppliers. Jumpers must be set to properly configure the board for the PC.

The drivers included with the COMM900 package are written to support the PCX ARCNET module.

The default software addresses used by the example programs are defined as follows:

Module I/O address	Module memory address	Mode node number
0x2e0	oxccooo	3

Installation

If you are configuring a host computer to communicate to a PiC via ARCNET, you will need to install an ARCNET board into the ISA bus of your computer. The card will require an I/O port and memory address to be selected. It will also require an available IRQ to be correctly installed. Please refer to the documentation that comes with your card for more information.

CAUTION

If the card is not installed correctly or if a conflicting address or IRQ is chosen, the communication software may run intermittently or not at all.

1.4 - Software requirements and installation

Software requirements and installation procedures for the COMM900 package are covered in this section.

Requirements

In the PiC
PiCPro/PiCServoPro - Version 7.1 or higher required

In the PC
MS DOS 6.22 or higher
Windows, Windows 95 or NT

Installation

All files included in this package start with the characters C_ to indicate communications. The 'C' and main ladder programs need not retain this convention. The ASFB ladders should retain the names as they were delivered to maintain compatibility with future offerings from Giddings and Lewis.

The COMM900 files are grouped into directories according to their usage (i.e., network, serial). You will use the files from the directory applicable to your communication setup. If you will be developing a network application, copy the files from the \NET_LDO\ASFBS directory to your ASFB directory. Example ladder programs can be copied from the \NET_LDO\EXAMPLES directory. If you will be developing a serial application, copy the files from the \SER_LDO\ASFBS directory to your ASFB directory. Example ladder programs can be copied from the \SER\EXAMPLE directory.

For more information on copying the files, please see the ASFB Guideline section at the beginning of this manual.

> **NOTE**
>
> If you copy both the serial and network ASFB files into your ASFB directory, you will see duplicate function warnings when you start PiCPro. This is because both versions share some function blocks and are included in both library files.
>
> Typically, this is not a problem unless the function block is changed and not updated in both libraries.

The program code for the PC drivers is found in the \NET_C or \SER_C directories for the network or serial versions respectively. Simply copy these files to the directory that you will be developing your application in.

The following files are included in the COMM900 ASFB package.

NET-C The files in this directory are the 'C' files for communicating over an ARCNET network.

N900.BAT	Example batch file for compiling.
C_CNET.C	Communication driver.
C_CNET.H	Include file used in compiling.
C_CNET.OBJ	Object module.
C_NET900.EXE	Executable example program using communication.
C_NET900.C	Source of example program.
C_NET900.OBJ	Object module.

NET-LDO The files in this directory are the PiC ladder files for communicating over an ARCNET network. They are in two directories.

ASFBs

C_MEMX.LDO*	ASFB for memory transfer (used in C_NETXCV.)
C_MEMX.REM*	Remark file.
C_NETXCV.LDO	ASFB for communication through the ARCNET port.
C_NETXCV.REM	Remark file.
C_NETLIB.LIB	Library of ASFBs.
C_NETXFR.LDO	ASFB for initiating a network message.
C_NETXFR.REM	Remark file.
C_BYTEMV.LDO*	ASFB for reading and writing memory.
C_BYTEMV.REM*	Remark file.
C_STRMOV.LDO*	ASFB for reading and writing STRING memory.
C_STRMOV.REM*	Remark file.

* These are auxiliary ASFB files that support the C_NETXCV and C_NETXFR ASFBs. Do not access these ASFBs in your ladder.

EXAMPLES

C_NET900.LDO	Main LDO example using the C-NETXCV function block.
C_NET900.REM	Remark file.
C_NETSND.LDO	Main LDO example using the C-NETXCV and C_NETXFR function blocks.
C_NETSND.REM	Remark file.

SER_LDO The files in this directory are the PiC files for serial transfers. They are in two directories.

ASFBs

C_MEMX.LDO*	ASFB for memory transfer used in C_COM900.LDO.
C_MEMX.REM*	Remark file.
C_SERLIB.LIB	Library that contains ASFBs.
C_SERXCV.LDO	Communications driver.
C_SERXCV.REM	Remark file.
C_BYTEMV.LDO*	ASFB for reading and writing memory.
C_BYTEMV.REM*	Remark file.
C_STRMV.LDO*	ASFB for reading and writing STRING memory.
C_STRMV.REM*	Remark file.

EXAMPLES

C_SER900.LDO	Main LDO.
C_SER900.REM	Remark file.

SERIAL-C The files in this directory are the 'C' files for serial transfers.

CS900. BAT ,	Example batch file for compiling ‘C’ files.
C_CSER.C	Serial communication driver for ‘C’ programs.
C_CSER.H	Include file used when compiling ‘C’ files.
C_CSER.OBJ	Object file.
C_SLIB2.OBJ	Object module.
C_SLIB2.ASM	Serial port hardware interface for ‘C’ programs.
C_SER900.OBJ	Object file.
C_SER900.EXE	Executable example ‘C’ program using communications.
C_SER900.C	Source of example ‘C’ program using communications.

* These are auxiliary ASFB files that support the C_SERXCV ASFB. Do not access these ASFBs in your ladder.

1.5 - Programming requirements

When programming the host computer to communicate to the PiC, it is important that both sides are configured correctly and do not conflict with each other. If there is a problem with the configuration, you will typically get one of the error messages shown in Appendix B of this manual.

The important thing to note if programming with serial communication drivers is that the serial port configurations are the same for the PiC and the PC. This includes the baud rate, parity, number of stop bits, etc.

If programming with the network communication drivers, make sure that each device (i.e., the PC and each PiC) have a different node number. It is also important that the settings for your ARCNET card be entered correctly into the network communications drivers. The software in the PC must know the settings of the ARCNET card in order to communicate to it.

1.6 - COMM900 startup

To start up COMM900 on the PiC and PC, follow the steps below for your application.

Network transfers

1. PiC- Download and scan the C_NET900 module.
2. PC - Run the C_NET900.EXE file.

Serial transfers

1. PiC - Download and scan the C_SER900 module.
2. PC - Run the C_SER900.EXE file.

Chapter 2 - Programming the PiC

2.1 - LDO programming overview

This chapter describes the ladder diagram software that executes in the PiC to support serial or network communications with a program executing in a PC. Either serial or network communication uses the same fundamental technique.

The main ladder diagram uses ASFBs that handle the communications task in the PiC. All command processing and error checking is handled by the ASFBs. ASFBs are functionally identical to UDFBs (User Defined Function Blocks). UDFBs are explained in the Software Manual.

The ASFBs process requests for data. These requests may originate from a host computer or another PiC. It is important to remember, however, that a PiC may send a message to another PiC but never to the host computer. The software running in the host does not support unsolicited messages.

1. Main LDO

   This is the user application program and contains the definition of the reserved data area within the PiC that will be accessed. The main LDO makes a function call to the ASFB that performs the communication function. The reserved data area is in the form of a rigid data structure. It must be defined as shown in the C-NETXCV ASFB description.

   The main LDO calls an ASFB that accesses the data structure and performs the communication function over the serial or ARCNET port. The main LDO can access the memory locations in the data structure directly or additional logic can be added to the LDO to copy that data to different variables.

2. ASFBs for communication

   The ASFBS are modules written by Giddings & Lewis to perform the data communication functions. The type of function blocks you use will depend on your application.

   PC to PiC Data Transfers

   C-NETXCV or C_SERXCV

   These are transceiver function blocks. They handle the hardware initialization and communications services for the main LDO. In addition, all protocol and frame encode/decode is done by these function blocks.

   PiC to PiC Data Transfers (Network Only)

   C-NETXCV and C_NETXFR

   The C_NETXFR function block is a transmitter function block. It is used to initiate a request for the C-NETXCV function block to send a message to another PiC.

Main LDO

The main ladder module requires the transceiver ASFB (C_NETXCV) to be incorporated into its logic. This function block must be enabled every scan and uniquely identifies a node on the ARCNET network. An example ladder module named C_NET900.LDO is included to illustrate how the function block is programmed. This program can be used to communicate to a host computer or to another PiC.

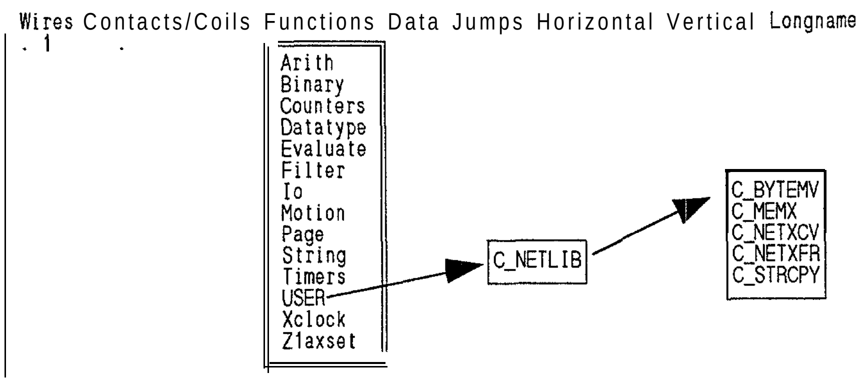
The main ladder module may have the optional transmitter ASFB (C_NETXFR) incorporated into its logic. This function block must be one-shot and allows a node to initiate a network message to another PiC node. An example ladder module name C_SND900.LDO is included to illustrate how the function block is programmed. This program can be used to send a message to another PiC. This ASFB cannot be used to send a message directly to a host computer.

NOTE: The C_NETXFR ASFB requires the C_NETXCV ASFB to operate.

The following diagram shows that when the main module is open and the Functions menu is displayed, the ASFBs library is found under USER.

The source LDOs for the ASFBs can be viewed by selecting User Function under View. Refer to the description of UDFBs in the Software Manual for further information.

Figure 2-1. NET900 ASFBs

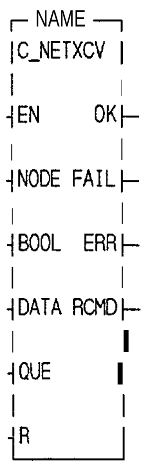


NOTE: The C_BYTEMV, the C_MEMX, and the C_STRCPY function blocks support the C_NETXCV and the C_NETXFR function blocks You do not use them in your application ladder but they must be in the ASFB library.

C_NETXCV

USER

Network Transceiver



Inputs: EN (BOOL) - enables execution (Typically enabled.)
NODE (USINT) - node number of this PiC
BOOL (ARRAY) - memory area for boolean data
DATA (STRUCT) - memory area for all data except boolean
QUE (ARRAY OF STRUCT) - Request to transfer message queue
R (STRUCT) - last received message header data

Outputs: **OK** (BOOL) - initialization complete
FAIL (BOOL) -initialization failed
ERR (INT) - error number
RCMD (BOOL) - message received

C_NETXCV is used to handle network data transfers between the PiC in which it is executed and a PC or between the PiC in which it is executed and another PiC. It performs the following functions:

1. Checks and initializes the communications daughter board if used.
2. Assigns a unique node (identification) number to this PiC.
3. Defines memory areas in the PiC for data that is transferred.
4. Performs the read or write operation on the memory area.
5. Sends a response to the PC or PiC indicating (un)successful completion of the command.

Inputs

EN The input at EN enables communications with the PC or another PiC when energized. De-energizing this input causes communications to stop being processed for this node and the communications port to be closed. This input should remain energized every scan. Typically, it is wired to the power bus.

NODE The input at NODE should be a number (1 - 255 excluding 65 or 41 hex) for this PiC. This number should not be assigned to any other device (PiC or PC). The number is used to uniquely identify this PiC on the ARCNET network.

IMPORTANT

If the PiCPro over ARCNET feature is used, this node number must match the node ID set in the PiCPro menu Process or - set Ne twork node ID. If it does not, initialization will fail. If the node number is unknown, a node number of zero (0) can be entered and the function block will automatically select the appropriate number.

B 0 0 L The input at BOOL should be an array (created and declared) with 2 to 999 boolean elements. This area is used to hold boolean data that is being transferred.

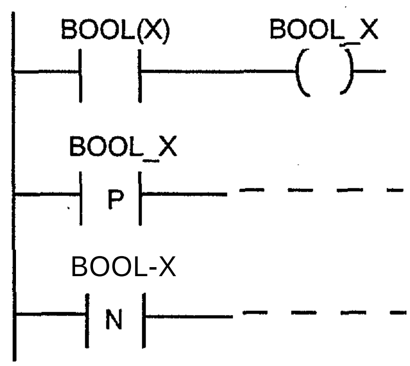
IMPORTANT

The boolean array size is also entered in the structure at the DATA input of C_NETXCV. The default size specified there is 2. If you create an array with a different size, be sure to change the default in the structure at the DATA input also.

IMPORTANT

Do not use a positive or negative transistional contact in your LDO with the BOOL array.

If it is necessary to set up a transistional contact with a BOOL array, use the BOOL array to energize another boolean coil. Then use this boolean for the transistional contact as shown in the example below.



DATA The input at DATA is used to specify the name of the main data area. This data area is a structure that you enter which contains every data type available except booleans. For each data type that exists in the PiC, there are two field entries in the structure. The format for this data structure is shown in Table 2-1.

IMPORTANT

The format of the structure has already been defined and must be followed. The data types must be entered exactly as listed in the declarations table. But the size of each data type is selected by you, with each type having a minimum size of 2 (the default) and a maximum of 999. Always be sure to change the I n i t Val column in the software declarations table when you change the size of a data type. Failure to do so will generate an error or cause invalid data transfer.

IMPORTANT

If your application does not require a math co-processor and you are running PiCPro Version 7.0 or higher, an error message stating that a math co-processor is required will appear when downloading a program that contains this data structure. This message is generated because the data structure has data types that require a math co-processor, i.e. REAL, LWORD, ULINT, etc. You can choose to ignore this error and continue downloading.

OR

If you want to eliminate the message from appearing, you can change the following datatypes in the DATA structure.

Name	Existing Data Type	Change to:
.LWORD_D	LWORD (0..1)	**DINT (0..3)***
.ULINT_D	ULINT (0..1)	**DINT (0..3)***
.LINT_D	LINT (0..1)	**DINT (0..3)***
.REAL_D	REAL (0..1)	**DINT (0..1)**
.LREAL_D	LREAL (0..1)	**DINT (0..3)***

The data types that require a math co-processor are now eliminated from the program and, consequently, the error message will not appear. The DINT arrays you are replacing the existing data types with do not require a math co-processor.

*The array size for these DINTs is double the original size for the 64-bit variables. This insures that the memory map for the data structure remains the same.

An example showing how to configure this data structure follows after the data descriptions.

Table 2-1. Structure at DATA Input to C_NETXCV

Name DATA	Type STRUCT	Init Val
.BOOL_S	UINT	2
.BYTE_S	UINT	2
.BYTE_D	BYTE(0..1)	
.WORD_S	UINT	2
.WORD_D	WORD (0..1)	
.DWORD_S	UINT	2
.DWORD_D	DWORD (0..1)	
.LWORD_S	UINT	2
.LWORD_D	LWORD (0..1)	
.USINT_S	UINT	2
.USINT_D	USINT (0..1)	
.UINT_S	UINT	2
.UINT_D	UINT (0..1)	
.UDINT_S	UINT	2
.UDINT_D	UDINT (0..1)	
.ULINT_S	UINT	2
.ULINT_D	ULINT (0..1)	
.SINT_S	UINT	2
.SINT_D	SINT (0..1)	
.INT_S	UINT	2
.INT_D	INT (0..1)	
.DINT_S	UINT	2
.DINT_D	DINT (0..1)	
.LINT_S	UINT	2
.LINT_D	LINT (0..1)	
.REAL_S	UINT	2
.REAL_D	REAL (0..1)	
.LREAL_S	UINT	2
.LREAL_D	LREAL (0..1)	
.STRING_S	UINT	2
.STRING_D	STRING [SO] (0..1)	
.DATE_S	UINT	2
.DATE_D	DATE (0..1)	
.DANDT_S	UINT	2
.DANDT_D	DANDT (0..1)	
.TIME_OF_DAY_S	UINT	2
.TIME_OF_DAY_D	TIME_OF-DAY (0..1)	
.TIME_S	UINT	2
.TIME_D	TIME (0..1)	
.CUST1_S	UINT	2
.CUST1_D	USINT (0..1)	
.CUST2_S	UINT	2
.CUST2_D	USINT (0..1)	
.CUST3_S	UINT	2
.CUST3_D	USINT (0..1)	
.CUST4_S	UINT	2
.CUST4_D	USINT (0..1)	
.CUST5_S	UINT	2
.CUST5_D	USINT (0..1)	
.CUST6_S	UINT	2
.CUST6_D	USINT (0..1)	
.CUST7_S	UINT	2
.CUST7_D	USINT (0..1)	
.CUST8_S	UINT	2
.CUST8_D	USINT (0..1)	
.CHKSUM	UINT	54321
	END_STRUCT	

Description

Size of the array at the previous input (BOOL) to C_NETXCV.

These members of the DATA structure define an array data area and its size for every data type except boolean.

The members work in pairs. The first entry defines the size of the array. Its name ends with "S". The second entry defines the array data area. Its name ends with "D".

The name of the structure and the size of any or all arrays can be changed. The order of the members of the structure and the data types *must not* be changed.

If the size of an array is changed, ensure that the initial value for its size variable is also changed.

Up to 240 bytes of data can be transferred in any one read or write operation. The number of array entries this represents depends on the data type and the number of bytes required to hold the data type.

The CUST members of the DATA structure hold the user's customized data.

As with the arrays for the data types, these members work in pairs. The difference is that the data member (i.e., CUST1_D) does not have to be an array. You may elect to replace the 'CUST1_D' entry with one or more of any of the data types. You must, however, count up the exact number of bytes used and put that number into the initial value column in the software declarations table for the size of your data area. See the example on the following page.

IMPORTANT

The last data variable CHKSUM must be included in the structure with the initial value as shown. This memory location with a known value is used by the ASFB to verify the configuration of the data found in the data structure, If the structure is not configured correctly, an error will be generated upon initialization.

Example

In Table 2-1, the DATA input structure from the software declaration table is shown with all the supported data types as different members of one large structure. Boolean data is also supported, but it is defined as its own array specified at the BOOL input of the function block.

The first step is to determine how many addresses of each of the different data types will be required for the application. The array size for the appropriate data type must then be modified to reflect the number of addresses required for each data type. All of the data types except boolean variables can be found within the DATA input structure and have a name ending with '_D" for Data memory. Boolean data is stored in a separate BOOL array specified at the BOOL input of the function block.

The second step is to define the current array size for each data array by changing the initial value of the size variable located above the data array entry. All of the size variables including the boolean array size can be found within the DATA input structure and have a name ending with 'S' for data memory Size. Enter in the initial value column for the size variable, the total number of array elements for that data type. For example, if the boolean array is defined to be BOOL(0..32) enter a 33 for the initial value for DATA.BOOL_S. If the integer array is defined to be DATA.INT_D(0..99) enter a 100 for DATA.INT_S.

NOTE: It is very important that the initial value for each of the data memory size variables accurately reflect the actual array size of each data variable.

The custom data area identified as CUST1 through CUST8 at the bottom of the data structure is used to access data that is better stored in format other than an array. Typically this would be used if you need to send multiple types of data in a single message transfer. With any data transfer, a maximum of 240 bytes can be sent per transfer.

The custom data area lets you replace the DATA.CUSTx_D entry in the data structure with any one or more data type variables. There is no limit to the number or type of variables, but they must all be inserted one after the other in the data structure immediately following the data memory size variable. The initial value for the size variable is then the total number of bytes consumed for the data types you entered.

If, for example, you want to transfer two INT variables and 20 REAL variables, the custom data area could be modified as shown below to transfer both sets of variables in one data transfer. Note that the value stored in the initial value column of the size variable represents the total number of bytes that your data consumes. Refer to the PiCPro Software Manual for the byte counts for each data type.

.CUST1_S	USINT	84	
.INT1	INT		(2 bytes)
.INT2	INT		(2 bytes)
.REAL	REAL(0..19)		(80 bytes)

The data structure allows for up to eight different custom data types. Each custom data type must be 240 bytes or less if they are sent in one transfer.

QUE The input at QUE should be the array of structures shown in Table 2-2. This input is the same as the QUE input to the C-NETXFR function block. The que communicates a data transfer request between the C_NETXFR and the C_NETXCV function blocks. The name of the array of structures can be changed. The format, member names, and data types *must not* be changed.

Table 2-2. Array of structures at QUE input to C_NETXCV and C-NETXFR

Name	Type	Description
	STRUCT(0..10)	
.NODE	USINT	Number of the source node (1 - 255).
.RSVD	USINT	Reserved for future use.
.CMD	USINT	Value = 1 if write command; value = 0 if read command.
.TYPE	USINT	Type of data being transferred. This identifies the array in the DATA structure See Appendix B for types.
.LNDX	UINT	Number of the local array element where data is to start being read from or written to.
.CNT	USINT	Number of elements being transferred The number of elements depends on the size (in bytes) of each element. Up to 240 bytes can be transferred.
.RNDX	UINT	Number of the remote array index wherr data is to start being read from or written to.
.STAT	USINT	Response status. This is a status returned from the remote node indicating if the message was processed successfully. See Appendix A for error codes.
	END-STRUCT	

R The input at R should be the predefined structure shown in Table 2-3. Command (header) data is placed in this structure by the PiC when a message is received and has been processed. The data is not required for data transfers. It is provided for programmers to reference and can be used to determine the content of the last message processed.

The name of the structure can be changed. The format, member names, and data types *must not* be changed.

Table 2-3. Structure at R input to C_NETXCV

Name	Type	Description
	STRUCT	
.NODE	USINT	Number of the source node (1 - 255).
.RSVD	USINT	Reserved for future use.
.DIR	USINT	Value = 1 if write command; value = 0 if read command.
.TYPE	USINT	Type of data being transferred. See Appendix B.
.LNDX	UINT	Number of the array element where data is to start being read from or written to (0 to n - 1, where n is the size of the array). The array is a member of the DATA structure. The member is specified by .TYPE above.
.CNT	USINT	Number of elements being transferred. The number of elements depends on the size (in bytes) of each element. Up to 240 bytes can be transferred.
. RSVD2	USINT	Reserved for future use.
	END-STRUCT	

Outputs

OK The output at OK will be set if the initialization has been completed successfully.

FAIL The output at FALL will be set if the initialization was not completed successfully. The output at ERR will indicate the error number.

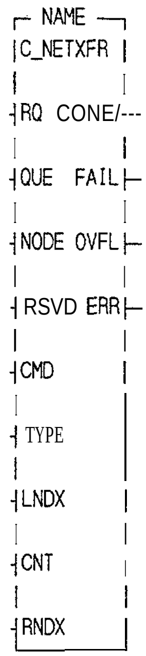
ERR The output at ERR does not equal 0 when an error occurs in' the initialization of the C_NETXCV ASFB.

ERR #	Description
1	The ARCNET hardware ID check failed.
2	The transmitter is not available. An ARCNET hardware failure has occurred.
3	The power-on reset flag cannot be cleared. An ARCNET hardware failure has occurred.
4	The number specified at NODE is assigned to another node. Check NODE numbers.
5-44	Check Appendix B in the Software Manual for errors.
133	Structure defined at DATA is invalid. Check Init Values declared for all entries in the structure defined in the software declarations table.
>1XXX	The node number has been set by PiCPro and is different than the number you entered at the NODE input. The XXX holds the PiCPro Network ID node number 001 through 255.

RCMD The output at RCMD is energized for one scan if a new message is received. The message header is available in the structure defined at the R input.

C_NETXFR
USER

Network Transmitter



Inputs: RQ (BOOL) - requests execution (Typically, one-shot)
QUE (ARRAY OF STRUCT) - memory area for queuing multiple requests. This memory is shared with the C-NETXCV function block.
NODE (USINT) - destination node number
RSVD (USINT) -reserved for future use
CMD (USINT) - command to be performed
TYPE (USINT) -data type to be transferred
LNDX (UINT) - local array index
CNT (USINT) - number of array elements to transfer
RNDX (UINT) - remote or destination array index

outputs: DONE (BOOL) - execution complete
FAIL (BOOL) -execution failed
OVFL (BOOL) - energized if more than 10 messages have been queued
ERR (INT) -0 if initialization is successful
0 if initialization is unsuccessful

The C_NETXFR ASFB is used to initiate a network data transfer message to another PiC. This ASFB must be used with the C-NETXCV ASFB. It performs the following functions:

1. Queues data transfer request to be processed by the C-NETXCV ASFB and sets the queue overflow output if the queue is full.
2. Waits for the message to be sent and sets the DONE or FAIL outputs depending on the status of the reply from the destination node.

Inputs

RQ The input at RQ should be one-shot to request a data transfer. The format of the command that will be initiated is defined by the other inputs to this function block. The RQ input is typically not transitioned again until after either the DONE or FAIL output has been sent from the previous request.

QUE The input at QUE should be the array of structures shown in Table 2-2. This input is the same as the QUE input to the C-NETXCV function block.

NODE The input at NODE should be the node number of the PiC that will receive the command.

RSVD Reserved for future use.

CMD The input at CMD defines the command to be sent.
1 = write data from this node to another node.
0 = read data from another node to this node.

TYPE The input at TYPE indicates the data type to be transferred. See Appendix B.

LNDX The input at LNDX indicates the starting local array index that data will be written from or read to by the data transfer (0 to n - 1, where n is the size of the array).

CNT The input at CNT indicates the number of array elements to READ/WRITE.

RNDX Number of the remote array index where data is to start being read from or written to (0 to n - 1, where n is the size of the array).

Outputs

DONE The output at DONE will be set when the command has been sent and received successfully by the destination node.

FAIL The output at FAIL will be set when the command was not sent or received successfully by the destination node. The output at ERR will indicate the error.

OVFL The output at OVFL will be set when more than 10 data transfer requests have been requested at one time. The output at ERR will indicate the error.

ERR The output at ERR does not equal 0 when an error occurs in the execution of C_NETXFR.

ERR #	Description
1	Unrecognized data type.
2	Requested data exceeded array size.
4	Data byte count > 240.
5	Destination nodes data structure is invalid. Check Init Values declared in the software declarations table.
17	Timeout waiting for response. Network may be busy. Resend message. If this does not work, check ladder scan and hardware wiring.
18	Not enough characters in response.
33	Queue overflow. More than 10 messages sent before the first one is complete.

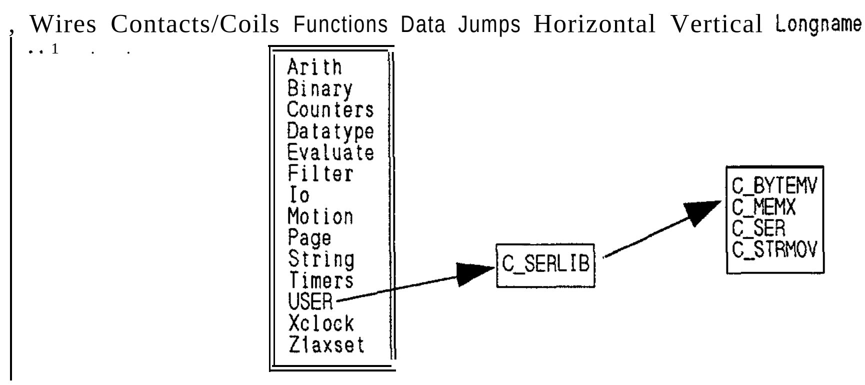
Main LDO

The main ladder module requires the transceiver C_SERXCV ASFB incorporated into its logic. This function block must be enabled every scan and accesses the serial port on the PiC. An example ladder module named C_SER900.LDO is included to illustrate how the function block is programmed. This program can be used to communicate to a host computer.

The following diagram shows that when the main module is open and the Functions menu is displayed, the ASFBs library is found under USER.

The source LDOs for the ASFBs can be viewed by selecting User Function under View. Refer to the description of UDFBs in the Software Manual for further information.

Figure 2-2. Serial ASFBs

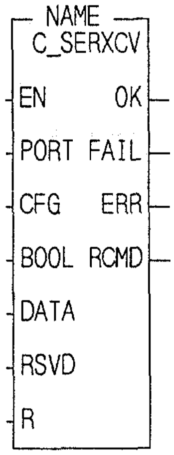


NOTE: The C_BYTEMV, C_MEMX, and C_STRMV function blocks support the C_SERXCV function block. You do not use them in your application ladder but they must exist in the ASFB library.

C_SERXCV

USER

Serial Communications



Inputs:
EN (BOOL) - enables execution (Typically enabled.)
PORT (STRING) - PiC serial port
CFG (STRING) - configuration of port
BOOL (ARRAY) - memory area for boolean data
DATA (STRUCT) - memory area for all data except boolean
RSVD - reserved for future use
R (STRUCT) - last received message header data.

Outputs:
OK (BOOL) - initialization complete
FAIL (BOOL) - initialization failed
ERR (INT) - error number
RCMD (BOOL) - message received

The C-SERXCV function block is used to make serial (RS-232) data transfers between the PiC in which it is executed and a PC. It performs the following functions:

1. Defines and configures the PiC serial port.
2. Defines memory areas in the PiC for data that is transferred.
3. Performs the read or write operation on the memory area.
4. Sends a response to the PC indicating (un)successful completion of the command.

Inputs

EN The input at EN should be energized to enable communications with the PC. De-energizing this input causes communications to stop being processed for this node and the communications port to be closed. This input should remain energized every scan. Typically, it is wired to the power bus.

PORT The input at PORT specifies which serial channel on the PiC is being used for communications with the PC.
If user port 2 on the PIC CPU module is to be used, create a string variable and initialize it as 'USER:$00'.
If a 2 or 4 channel serial communications module is used, define the port with the ASSIGN function block (described in the Software Manual). The ASSIGN function block must execute before this input can be used for the serial communications module.

C FG The input at CFG establishes the communication parameters for the PiCs serial port. Create and assign a string at this input in the same format that is defined for the CFGZ input to the CONFIG function block (explained in the Software Manual).
Ensure that the parameters defined here match the parameters defined for the PC (in the SERIAL-OPEN function).
NOTE: Drivers in the PC do not recognize XON/XOFF. Use hardware handshaking for 19200 or greater baud rates.

BOO L The input at BOOL should be an array (created and declared) with 2 to 999 boolean elements. This area is used to hold boolean data that is being transferred.

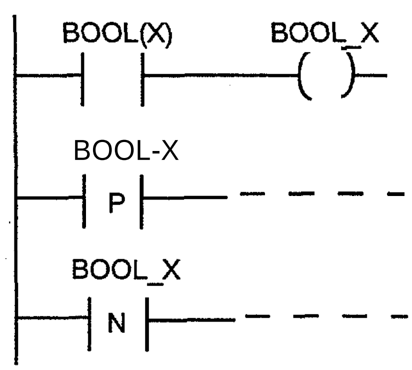
IMPORTANT

The boolean array size is also entered in the structure at the DATA input of C_SERXCV. The default size specified there is 2. If you create an array with a different size, be sure to change the default in the structure at the DATA input also.

IMPORTANT

Do not use a positive or negative transitional contact in your LDO with the BOOL array.

If it is necessary to set up a transitional contact with a BOOL array, use the BOOL array to energize another boolean coil. Then use this boolean for the transitional contact as shown in the example below.



DATA The input at DATA is used to specify the name of the main data area. This data area is a structure that you enter which contains every data type available except booleans. For each data type that exists on the PiC, there are two field entries in the structure (except booleans). The format for this data structure is shown in Table 2-l in the network transfer section.

IMPORTANT
The format of the structure has already been defined and must be followed. The data types must be entered exactly as listed in the declarations table. But the size of each data type is selected by you, with each type having a minimum size of 2 (default) and a maximum of 999. Always be sure to change the Init Val column in the software declarations table when you change the size of a data type. Failure to do so will generate an error or cause invalid data transfer.

RSVD Reserved for future use.

R The input at R should be the same as defined for the R input to the C_NETXCV ASFB. It is the predefined structure shown in Table 2-3 in the network transfer section.

Outputs

OK The output at OK is set if the initialization completed successfully.

FAIL The output at FAIL is set if initialization was not completed successfully. The output at ERR will indicate the error.

ERR The output at ERR will be 0 when an error occurs in the initialization of the C_SERXCV ASFB.

ERR #	
1 - 44	Check Appendix B in the PiCPro Software Manual for a description of these errors.
133	The structure defined at DATA is invalid. Check Init Values declared for all entries in the structure defined in the software declarations table.

RCMD The output at RCMD is energized for one scan if a message is received. The message header is available in the structure defined at the R input.

Chapter 3 - Programming the Host Computer

3.1 - Overview

This chapter explains how a host computer (personal computer or PC) is programmed or configured to communicate to the PiC900 family of controls Several different drivers will be discussed. Each driver has application programming requirements, but all implement the standard Giddings & Lewis COMM900 communication protocol. This means that the PiC must be running either the serial (C_SERXCV) or network (C NETXCV) transceiver function blocks to communicate to the PC. These function blocks are described earlier in the manual and come with the COMM900 ASFB software package.

Regardless of which communication driver is used in the PC, the main application ladder running in the PiC must incorporate either the serial or network transceiver function block.

3.2 - Interfacing to Standard COMM900 Device Drivers

COMM900 provides a function level interface for both serial and network communications to a PiC. The main application program is typically written in 'C' or 'C++' and "calls" the functions provided to handle the task of communicating between the PC and the PiC(s). The serial and network drivers are written to support the DOS operating system only. Applications developed using these drivers must be run in DOS.

3.3 - Network Drivers

When performing network communications, the main application program must interface to three different functions. These functions perform the initialization and data communication functions over ARCNET.

ARCINIT The ARCINIT function is called once when the application is started. It will initialize the ARCNET hardware to the selected hardware settings and will make this device an active node on the network.

ANYBODY The ANYBODY function may be called at any time but is typically called once when the application is started to query the network to identify if there is at least one other node on the network.

NET900 The NET900 function is called to initiate a data transfer message with any PiC node on the network. It will transfer the message and verify through a response mechanism that the message was transmitted and received successfully.

Each of these functions will be explained in the next section.

ARCINIT (p1 ,p2,p3,p4)

The ARCINIT function is called to initialize the ARCNET hardware in the PC.

> **IMPORTANT**
>
> The settings shown below must match those selected in order for the ARCNET board to operate correctly with your PC configuration. If these settings do not match, communication will fail.

Parameters

Parameter	Default Value	Description	Ctt Data Type
p1	2e0 Hex	I/O Port Address	short int
p2	cc00 Hex	Memory Base Address	short int
P3	00 Hex	Memory Offset	short int
P4	03 Hex	Node Number	short int

Return Status

A value of zero (0) indicates that the ARCNET board initialized successfully. Other error codes are described in Appendix A. The C++ return value data type is int.

Example

```
/* Initialize the ARCNET hardware */

init_ok=arcinit(0x2e0,0xcc00,0x0,0x03);

if (init_ok !=O){

        /*Error initializing ARCNET hardware */

        printf("ARCNET initialization failure, error #%\n",init_ok);

}
```

ANYBODY ()

The ANYBODY function is called to query the network to see if there is another node on the network. It should be called at least once at the start of the application program.

Parameters

None

Return Status

A value of zero (0) indicates that the network is operating and that there is at least one (1) other node communicating on the network. Other error codes are described in Appendix A. The C++ return value data type is int.

Example

```
/* Check network for other nodes */

other nodes =anybody();

if (other-nodes ! =0) {

        /*No other ARCNET nodes found*/

        printf("No other nodes found error #%\n",init_ok);

}
```

NET900(p1,p2,p3,p4,p5,p6,p7,p8)

The NET900 function is called to initiate data communications between the PC and a PiC. It is used to read data from or write data to a PiC via an application program running in the PC. The data transferred can be any of the data types listed in Appendix B. Only one data type can be transferred per message. Multiple data types can be transferred using the custom data type. Each message can contain up to 240 bytes of data.

Parameters

Parm	Range of Values	Description	Ctt Data Type
p1	80 Hex (read) c0 Hex (write)	**Function** Command to perform READ or WRITE	unsigned char
p2	00 Hex thru lb Hex	**Data Type** Type of data to be transferred. See Appendix B.	unsigned char
P3	memory address	**Data Pointer** Pointer to memory area in PC where the data is stored. This pointer is defined as void allowing any data type to be passed.	unsigned char far *
P4	0 to 998	**Data Array Index** This index specifies the starting array element in the PiC identified in parameter 6.	unsigned int
p5	1 to 240 for all one byte data types 1 to 120 for all two byte data types 1 to 60 for all four byte data types 1 to 30 for all eight byte data types 1 for all string data types	**Data Quantity** Quantity of the array elements to transfer. The maximum number depends on the data type being transferred. Refer to the PiCPro Software Manual to determine the size for each data type.	int
P6	1 to 255 (excluding 65)	**Node** Destination node number	unsigned char
p7	1 to 3	**Retry Count** Number of times to retry sending message before reporting an error	unsigned char
P8	00 Hex or 01 Hex	**Miscellaneous Flags** Typically set to 00 Hex. Set to 01 Hex if the calling program does not support one byte data types or if you are using Visual Basic.	unsigned int

Return Status

A value of zero (0) indicates that the message was sent and received correctly. Other error codes are described in Appendix A. The C++ return value data type is int.

Example

```
#define READ-CMD 0x80          /*Define symbol to be hex 80*/
#define WRITE-CMD 0xC0         /*Define symbol to be hex CO*/
#define TYPE_INT 0x0A          /*Define symbol to be hex 0A*/

int int_data[5];               /*Declare array of integers*/


/*Send a READ message to node 6-Read 5 SINT variables starting at array index 3.*/

/*Retry once upon a failure.*/

comerr=net900(READ_CMD,TYPE_INT,&int_data[0],3,5,6,1,0);

if(comerr){

        /*Unable to send message or invalid message*/

        printf("Error sending message to PiC node #6, error #%d\n",init_ok);

}
```

3.4 - Serial Drivers

When performing serial communications, the main application program must interface to three different functions. These functions open, configure, and close the selected serial **port** and performs the data communication functions over the serial line.

SERIAL-OPEN The SERIAL OPEN function is called once when the application is-started. This function will open the serial port and configure it to the selected communication settings.

SER900 The SER900 function is called to initiate a data transfer message with the PiC node connected via the serial link. This function will transfer the data and verify through a response mechanism that the message was transmitted and received successfully.

SERIAL-CLOSE The SERIAL-CLOSE function is called once when the application no longer needs to communicate to the serial port. This function will close the serial port. The application will need to re-open the port using the SERIAL-OPEN function to communicate to the PiC once the SERIAL-CLOSE function is executed.

These functions will be explained in the next section.

SERIAL-OPEN (p1 ,p2,p3)

The SERIAL-OPEN function is called once at the beginning of your application to open and configure the serial port that will be used for communications to the PiC. Once opened this port cannot be used by any other programs.

Parameters

Parm	Range of Values					Description	C++ Data Type
p1	1 to 2					COM port number	unsigned char
p2	Bit 7 Hardware Hand-shaking 0=none 1=CTS/RTS	Bit 6, 5 N/A	Bit 4, 3 Parity 0,0=none 0,1=odd 1,1=even	Bit 2 Stop Bits 0=1 1=2	Bit 1, 0 Word Length 1,0=7 bit 1,1 = 8 bit	Configuration Byte Bit pattern to set port characteristics. **Important:** These must match settings specified at the CFG input of the C_SERXCV function block.	int
Example: 03 Hex = 00000011 Binary = 8 data bits, 1 stop bit, no parity, w/ no hardware handshaking.							
p3	1200, 2400, 4800, 9600, 19200					Baud Rate **Important:** This must match baud rate specified at the CFG input of the C_SERXCV function block.	unsigned int

Return Status

A value of zero (0) indicates that the serial port opened and configured successfully. Other error codes are described in Appendix A. Note that a successful status returned from this function does not guarantee correct pinout of the serial cable. The C++ return value data type is int.

Example

```
#define CONFIG_STR 0x03 /*8 bit, no parity, 1 stopbit, no handshaking*/
open_ok=serial_open(1,CONFIG_STR,9600);
if(open_ok!=0){
        /*Unable to open and configure serial port */
        printf("Error initializing serial port, error #%\n",open_ok);
}
```

SER900(p1,p2,p3,p4,p5,p6,p7,p8)

The SER900 function is called to perform data communications between the PC and a PiC. It is used to read data from or write data to a PiC via an application program running in the PC. The data transferred can be any of the data types listed in Appendix B. Only one data type can be transferred per message. Each message can contain up to 240 bytes of data. Multiple data types can be transferred using the custom data type.

Parameters

Parm	Range of Values	Description	C++ Data Type
p1	c0 Hex (write) 80 Hex (read)	**Function** Command to perform READ or WRITE	unsigned char
P2	00 Hex thru lb Hex	**Data Type** Type of data to be transferred. See Appendix B .	unsigned char
p3	memory address	**Data Pointer** Pointer to memory area in PC where the data is stored. This pointer is defined as void allowing any data type to be passed.	unsigned char far *
P4	0 to 998	**Data Array Index** This index specifies the starting array element in the PiC identified in parameter 6.	unsigned int
p5	1 to 240 for all one byte data types 1 to 120 for all two byte data types 1 to 60 for all four byte data types 1 to 30 for all eight byte data types 1 for all string data types	**Data Quantity** Quantity of the data array to transfer. The maximum number depends on the data type being transferred. Refer to the PiCPro Software Manual to determine the size for each data type.	int
p6	1 to 2	**Port** COM port number	unsigned char
p7	1 to 3	**Retry Count** Number of times to retry sending message before reporting an error	unsigned char
p8	00 Hex or 01 Hex	**Miscellaneous Flags** Typically set to 00 Hex. Set to 01 Hex if the calling program does not support one byte data types or if you are using Visual Basic.	unsigned int

Return Status

A value of zero (0) indicates that the message was sent and received correctly. Other error codes are described in Appendix A. The C++ return value data type is int.

Example

```
#define READ-CMD 0x80          /*Define symbol to be hex 80*/
#define WRITE-CMD 0xC0         /*Define symbol to be hex CO*/
#define TYPE-INT 0x0A          /*Define symbol to be hex 0A*/

int int_data[5];               /*Declare array of integers*/


/*Send a READ message to COM 1*/
/*Read 5 SINT variables starting at array index 3.*/
/*Retry once upon a failure.*/
comerr=ser900(READ_CMD,TYPE_INT,&int_data[0],1,5,6,1,0);
if(comerr)(
        /*Unable to send message or invalid message*/
        printf("Error sending message to PiC, error #%d\n",comerr);
    }
```

SERIAL-CLOSE (pl)

The SERIAL-CLOSE function is called once upon the completion of your application program to close the serial port. This will return the port to the operating system for use by other programs.

Parameter

Parameter	Range of Values	Description	C+t Data Type
PI	1 to 2	COM port number	unsigned char

Return Status

A value of zero (0) indicates that the port was closed successfully. Other error codes are described in Appendix A. The C++ return value data type is int.

Example

```
comerr=serial_close(1);
```

3.5 - Interfacing to Optional DLL Drivers

An option to the standard drivers that come with the COMM900 ASFB package is the PiC Dynamic Link Library (DLL) for Windows. This product is an add-on option for the COMM900 ASFB product and is available for both serial and network configurations. Microsoft Windows 3.1 is required when using the 16-bit DLL. Window 95 or NT is required when using the 32-bit DLL.

The software programming interface to the functions found within the DLL is identical to that described in the section entitled "Interfacing to Standard COMM900 Device Drivers" of this chapter. All function names and the calling parameters are the same, however, the underlying software has been modified to allow it to run under the Windows environment.

Whereas the standard device drivers that come with COMM900 are distributed in the form of individual object files (i.e., .OBJ files), these device drivers are distributed in Dynamic Link Libraries or DLLs.

A library is a mechanism by which multiple functions can be grouped and stored as a single file. Your application can then be compiled and linked with the library file to create the program the operator will run. The size of the resulting program then includes all of the main program plus the functions that were in the library file.

A DLL is simply another type of library. A DLL, however, is a special type of library supported by Windows allowing your application program to be compiled and linked without including the library file. This will keep the size of your program much smaller and Windows will automatically load the code found in the library when a function in that library is called in the main program. The main program must, however, register the names of all DLLs and the functions stored in each library upon startup so Windows knows where the code for each function is located.

Information on interfacing to a DLL can be found in the documentation for the programming language you are using.

3.6 - Interfacing to Optional DDE Server

Another option to the standard drivers that come with the COMM900 ASFB product is the Dynamic Data Exchange (DDE) server. This is an add-on option for the COMM900 ASFB product for the PiC. Microsoft Windows 3.1 is required when using 16-bit DDE. Window 95 or NT is required when using 32-bit DDE.

A DDE server is a Windows program that incorporates the software drivers required to communicate to the PiC900 family of controls using either serial or network communications. The software is configurable to any communications port or network configuration. The program can be run manually or placed in the Startup group for Windows to automatically run when Windows is started.

Any DDE client compatible Windows program can communicate to the DDE Server using a standard DDE message. A DDE message uniquely identifies the PiC node and memory address to be accessed. Any of the standard PiCPro data types can be used excluding the 64-bit data types.

The DDE server interfaces to the COMM900 transceiver function block. As with the standard device drivers, the DDE server can access up to 999 different addresses for each of the supported data types. To access any memory location, use the following DDE message syntax.

(application name}|topic!item

where,

'application name' is the program name PIC_DDE
'topic' is the node name assigned to the PiC
'item' is the memory address

A memory address consists of the short data type name (i.e. USINT for unsigned short integer) followed immediately by an array index number. Index numbers start at zero and go to 998. These memory addresses map directly to the data arrays found inside the structure specified at the DATA input to the C_NETXCV and C_SERXCV function blocks.

Example

In this example, the DDE server is used to access the tenth unsigned double integer value from the PiC node named 'CUTTER'.

PIC_DDE|CUTTER!UDINT9

This DDE address maps directly to DATA.UDINT_D(9) in the PiC control.

Refer to the PiC DDE Server User Manual for more information on the configuration and use of this product.

3.8 - Interfacing to Third Party Drivers

Another option to the standard drivers that come with the COMM900 ASFB product is to purchase a driver developed by a third party software developer to be used with an operator interface program or device. Giddings & Lewis works with many third party developers to incorporate the standard COMM900 protocol into their application software or hardware. These drivers are written to interface to the standard transceiver function blocks described in this manual. You will need the COMM900 ASFB software to get the function blocks to communicate with the third party driver.

Most third party software drivers implement both the serial and network interface to the PiC and can read or write virtually all of the standard PiCPro data types excluding the 64-bit data types. Within each data type there are typically up to 999 different memory locations that can be addressed.

The actual implementation of the device driver will vary from vendor to vendor. The software vendor should be contacted directly for any application questions or for device driver specifications or support.

NOTES

Appendix A - Error Codes

The following are common error codes returned by the communication drivers provided to communicate to a PiC from a host computer.

Error Codes from Local Node

Serial or network errors

ERR #	Description
-1	Invalid parameter in NET900 or SER900 function
-2	Received sequence number does not match command sequence number
-3	Response status error
-4	Response checksum error
-5	Timed out waiting for response - Network may be busy. Try to resend message.

Network errors only

ERR #	Description
-6	Cannot process command - network in reconfiguration
-7	Transmitter not available for use
-8	Timed out on transmit
-9	No other nodes on network - token not found
-10	NIM initialize command timed out without getting D1
-11	Incorrect status found after reset
-12	Power on reset flag could not be cleared
-15	Invalid port
-20	Invalid host mode
-21	Memory allocation error
-22	Driver initialization error

Error Codes from Remote Node

These are the six least significant bits of the STATUS byte in a response.

ERR #	Description
1	Unrecognized data type
2	Requested data exceeds array size
4	Data byte count > 240 bytes
5	Invalid data structure format
	Destination nodes data structure is invalid. Check Init Values declared in the software declarations table.

NOTES

Appendix B - COMM900 Data Types

When entering the type of data to be transferred, use the hex code or the label shown below. NOTE: When hex codes are entered as constants in PiCPro, they must be proceeded by 16#. For example, enter the hex value OA as 16#0A.

HEX	LABEL	Size of Data Type (in bytes)
00	TYPE-BOOL	1
01	TYPE-BYTE	1
02	TYPE-WORD	2
03	TYPE-DWORD	4
04	TYPE-LWORD	8
05	TYPEJSINT	1
06	TYPE_UINT	2
07	TYPE-UDINT	4
08	TYPE-ULINT	8
09	TYPE-SINT	1
OA	TYPE-INT	2
OB	TYPE-DINT	4
oc	TYPE-LINT	8
OD	TYPE-REAL	4
OE	TYPE-LREAL	8
OF	TYPE-STRING	Variable (2 bytes + string length)
10	TYPE-DATE	2
11	TYPE-DATETIME	4
12	TYPE-TIMEOFDAY	4
13	TYPE-TIMEDURA	4
14	TYPE_CUST1	User-definable
15	TYPE_CUST2	User-definable
16	TYPE-CUST3	User-definable
17	TYPE-CUST4	User-definable
18	TYPE_CUST5	User-definable
19	TYPE-CUST6	User-definable
1A	TYPE-CUST7	User-definable
1B	TYPE-CUST8	User-definable

NOTES

Appendix C - Protocol and Frame Definitions

This information is included for reference only. The user does not need to understand the protocol to use the package. All protocol encode and decode is handled within the supplied functions.

This protocol is to be used on a full duplex, asynchronous RS-232 serial connection between a Giddings & Lewis PiC and a PC. This protocol also applies to an ARCNET connection with the addition of a lo-byte header attached to the start of the message. Since the ARCNET connection can be used by PiCPro and/or the ladder via the COMM900 product, a header frame is required to designate to the transport layer in the PiC where an incoming message should be delivered.

PC to PiC communications is based on a Command/Response protocol. Only two types of messages are defined: Command messages and Response messages. This protocol uses a variable size frame with a fixed size header and trailer. The header and trailer account for the overhead of each message. All messages have 14 bytes of overhead: 11 in the header and 3 in the trailer, plus data bytes.

NOTE: Some messages will be all overhead. Example: Read command messages and Write response messages.

The PC is defined as the Master which initiates all communications. The PiC is defined as the Slave which will respond only to COMMANDS from the Master. The PC (Master) generates a COMMAND to the Slave and waits for a RESPONSE. When the RESPONSE is received the transaction completes, and in the case of a valid READ command, any data returned from the SLAVE is written to PC memory. ,

All transactions must be contained within one frame. Multi-frame transactions are not supported. For example, to read 100 Double Integers (DINTS) requires 400 bytes of data. Since a frame is allowed only 240 bytes of data, two reads are necessary to retrieve all 400 bytes.

If a RESPONSE message corresponding to the COMMAND sent is not received before the timeout timer expires, the COMMAND message is re-sent and the timer is re-started. A 'retry counter is incremented to reflect the retry and checked against the maximum limit. If the maximum limit for retries is reached, an error is returned.

With the release of PiCPro 7.0, two changes in the ARCNET protocol were required. These changes to the protocol only exist when you set up a PiCPro Network ID to be non-zero. If the ID is zero, the old protocol is run.

1. A message number (Byte #2 in the ARCNET Header Frame) was added. This byte was previously not used. The message number is incremented from 1 to 255 and rolls over to 1 after the first 255 messages. The message number is used to detect duplicate messages.

2. An acknowledgment (ACK) message must be sent for every message received. The ACK message indicates that the message was delivered successfully and a RESPONSE message is imminent. The ACK message consists only of the 10 byte ARCNET Header Frame for the message that was sent with Byte #1 changed to a value FE hex indicating a Ladder Acknowledgment. If the receiving node fails to acknowledge a message received within 200 ms the sender will resend the same message.

PiC Data Communications Frame Description

Serial Communications Frame

Command/Response Frame		
Command Header (11 Bytes)	Data	Trailer (3 Bytes)

ARCNET Communications Frame

	Command/Response Frame		
ARCNET Header (10 Bytes)	Command Header (11 Bytes)	Data	Trailer (3 Bytes)

ARCNET Header

Byte 0	Byte 1	Byte 2	Bytes 3, 4, 5, 6, 7, 8, 9
Vendor ID D7 Hex	Message Type	Message Number	Not Used

Vendor ID

D7 Hex	Giddings & Lewis ID

Message Type

01 Hex	Ladder Message
02 Hex	System Message
FE Hex	Ladder Acknowledgment
FF Hex	System Acknowledgment

Message Number

Uniquely identifies ARCNET message. Starts at one end and increments to 255 and then rolls back to 1.

Command\Response Frame

Command or Response Frame

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	
Start Code FO Hex	Source Node	Destination Node	Frame Byte Count	LSB Sequence Number	MSB Sequence Number	→

Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Data 0	
Control Word # 0	Control Word # 1	Control Word # 2	Control Word # 3	Control Word # 4	Data Byte # 0	→

Data 1	Data2	Data n	Byte n+12	Byte n+13	Byte n+14
Data Byte # 1	Data Byte # 2	Data Byte # n	Checksum LSB	Checksum MSB	Stop Code 81 Hex

Command - Control word definitions

Byte 6 **Control Word 0** Command Code -
CO hex - WRITE command
80 hex - READ command

Byte 7 **Control Word 1** Data type - See Data type code list in Appendix B.

Byte 6 **Control Word 2** **Data** byte count.

Byte 9 **Control Word 3** PiC Destination Index (low byte).

Byte 10 **Control Word 4** PiC Destination Index (high byte).

Response - Control word definitions

Byte 6 **Control Word 0** Response Code - same as the Command without the MSB
40 hex - WRITE! Response
00 hex - READ Response

Byte 7 **Control Word 1** Data type - See Data type code list in Appendix B.

Byte 6 **Control Word 2** Data byte count.

Byte 9 **Control Word 3** Status byte - See Status byte description following.

Byte 10 **Control Word 4** Reserved - not defined

Command/Response Code (Control Word 0)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CMD/RSP	WRITE/READ	Not Used	Not Used	Not Used	Not Used	Not Used	Not Used

Bit 7: Set to one for a command message. Set to zero for a response.

Bit 6: Set to one for a WRITE command. Set to zero for a READ command or response.

Bit 5:
. \
. Not used.
. /
Bit 0:

Response Status Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ERROR	Not used	E-code b5	E-code b4	E-code b3	E-code b2	E-code bl	E-code b0

Bit 7: If bit 7 is set, an error occurred in the message transaction.

Bit 6: Not used - set to 0.

Bit 5:
. \
. Contains an error code when bit 7 indicates an error exists.
. /
Bit 0:

Possible status values in a Response Frame (Control Word 3)

00 Hex No error

81 Hex Unrecognized Data Type (See Data Types in Appendix B.)

82 Hex Requested Data Transfer Exceeded Size of Array in the PiC

84 Hex Data Byte Count Exceeded Frame Size: > 240 bytes.

85 Hex Invalid configuration of the input data structure.

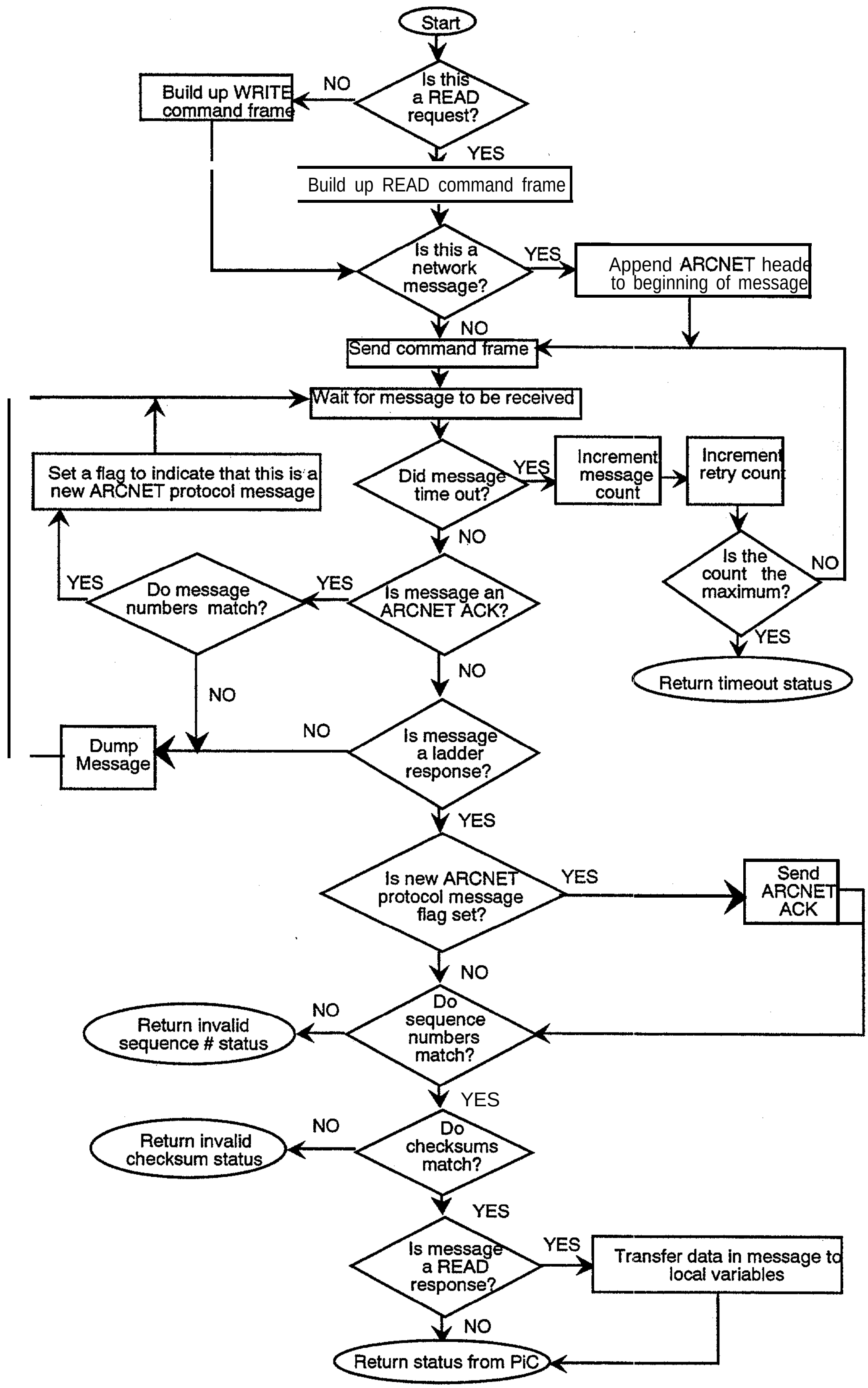
The checksum is generated by summing all the bytes, from the start byte to the byte before the checksum bytes, and then taking the 2s complement of that value. The checksum is a 16-bit value, with the LSB being sent first followed by the MSB.

Shown below is an example of a WRITE Command to send one LONG INT ('C' 4 byte number) from the PC to PiC:DATA.DINT_D(5). Also shown is an example RESPONSE to the WRITE command.

	WRITE			RESPONSE	
BYTE	DATA	Description	BYTE	DATA	Description
0	FO	start code	0	FO	start code
1	00	source node	1	01	source node
2	01	destination node	2	00'	destination node
3	12	frame byte count	3	OE	frame byte count
4	01	LSB sequence #	4	01	LSB sequence #
5	00	MSB sequence #	5	00	MSB sequence #
6	co	write command	6	40	response
7	OB	data type = DINT	7	OB	data type = DINT
8	04	# of data bytes	8	04	# of data bytes
9	05	dest address - low byte	9	00	status byte - no error
10	00	dest address - high byte	10	00	not used
11	00	data (LSB)	11	B1	LSB checksum
12	7F	data	12	FE	MSB checksum
13	FF	data	13	81	stop code
14	FF	data (MSB)			
15	AB	LSB checksum			
16	FB	MSB checksum			
17	81	stop code			

The following flowchart illustrates the command and response communication protocol.

Figure 1.1 Flowchart of data transfer steps



Index