

Version 9.X

SOFTWARE MANUAL

PiCPro™

GIDDINGS & LEWIS, INC.

Version 9.X

SOFTWARE MANUAL

PiCPro™

GIDDINGS & LEWIS, INC.

NOTE

Progress is an on going commitment at Giddings & Lewis. We continually strive to offer the most advanced products in the industry; therefore, information in this document is subject to change without notice. The illustrations and specifications are not binding in detail. Giddings & Lewis shall not be liable for any technical or editorial omissions occurring 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.

401-54513-00

Version 2298

© 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998 Giddings & Lewis, Inc.

Inductosyn is a registered trademark of Farrand Industries, Inc.

IBM is a registered trademark of International Business Machines Corp.

Pentium and Pentium Pro are trademarks of Intel Corporation.

Windows 95, Microsoft® and MS-DOS® are registered trademarks of Microsoft Corporation.

ARCNET® is a registered trademark of Datapoint

PiC900, PiCPro, PiCServoPro, PiCTune, PiCProfile, LDOMerge, PiCMicroTerm, and PiC Programming Pendant are registered trademarks of Giddings & Lewis, Inc.

Table of Contents

Introduction to the Software Manual	1-1
CHAPTER 1-Getting Started with PiCPro for Windows	
Introduction to PiCPro for Windows	1-3
The Computer Workstation	1-4
Hardware Connections	1-5
PiCServoPro/PiCPro Installation	1-6
Installing PiCServoPro on the hard disk	1-6
Using PiCServoPro/PiCPro	1-10
Menu Bars, Menus and Pop-Ups	1-12
Setting Up the Workstation Software	1-14
Workstation menu	1-14
Configuring the Printer	1-16
Printing Extended ASCII Characters	1-17
Project Manager	1-18
Introduction	1-18
Using Project Manager	1-19
Subdirectory Structures.....	1-26
CHAPTER 2- Introduction to PiCPro and Programming	
Introduction	2-1
Prerequisites for PiCPro Programming	2-1
Components of a Module	2-1
Networks	2-1
Software Declarations Table	2-4
Hardware declarations table	2-5
PiCPro menu system	2-7
Main menu	2-7
Module Menu	2-8
Network Menu	2-9
How to Create or Open a Module	2-10
How to add, change, and delete network elements	2-15
Cutting and pasting within a network	2-16
How to exit a network	2-16
How to create a second network	2-16
How to save and close a module	2-17
Designing networks	2-18
CHAPTER 3-Programming and Declarations	
Network Menu - Wires	3-2
Network Menu - Contacts/Coils	3-3
Network Menu - Functions	3-4
Chaining Functions/Blocks	3-9
Network Menu - Data	3-10

Bitwise	3-11
Numeric	3-12
String	3-13
Time:time of day	3-14
Time:duration	3-15
Function/Block Data	3-15
Data Groups of Variables - Arrays and Structures	3-16
Network Menu - Horizontal	3-19
Network Menu - Vertical	3-19
Network Menu - Longname	3-19
Module Menu - Declarations	3-20
Hardware Declarations - Master and Expansion racks	3-20
Software Declarations	3-23
Software Declarations - Variable Menu	3-27
Declaring Arrays and Structures	3-34
Arrays	3-34
Structures	3-37
Structure with Arrays	3-38
Array of Structures	3-40
Array of Structures with Arrays	3-42
Module Menu - View	3-52

CHAPTER 4-System Integration

Introduction	4-1
Module Menu - Download	4-2
Module Menu - Monitor	4-5
Monitor Menu - Animate ON	4-5
Monitor Menu - View ON	4-7
Monitor Menu - Edit View List	4-8
Module Menu - Print	4-13
Module Menu - UDFB Build (User defined function block)	4-16
Module Menu - Task Build	4-17
Module Menu - Hex-86 Dump	4-18
Module Menu - Bin-86 Dump	4-18
Module Menu - Build Dependency List	4-18
Processor Menu - Scan Control	4-21
Processor Menu - Flash Disk Operations	4-25
Processor Menu - Backup User Prog.	4-26
Processor Menu - Restore User Prog.	4-26
Processor Menu - Time Read/Set	4-26
Peer-to-Peer/PiC(Servo)Pro Communications	4-27
Processor Menu - Set Network Node ID	4-28
Processor Menu - Connect to Node	4-28
Processor Menu - Status	4-29
Troubleshooting an Application	4-30

CHAPTER 5-Motion Control Concepts

Background Information on Closed Loop Control	5-1
Introduction to Motion Control with the PiC	5-2
Basic Motion Control Programming Process	5-6

CHAPTER 6-Servo Setup and Tuning

Introduction to Servo Setup Data	6-1
Entering Servo Setup Data	6-4
Making a Setup Data Function/Saving a .SRV File	6-23
Creating a Servo Library for User-Defined Functions	6-23
Introduction to Tuning	6-25
Axis Tuning	6-26
Changing Inputs	6-27
Viewing Outputs	6-28
Other Features of the Servo Setup Program	6-30
Other Features of the Axis Menu	6-32

CHAPTER 7-Axis Profile Setup

Background Information on Ratio Profiles	7-1
Introduction to the PiC Profile program	7-5
Entering Profile Data	7-6
Making a Profile Function/Saving a .PRO File	7-14
Using the Profile Function	7-15
Other Features of the PiC Profile Program	7-16

CHAPTER 8-UDFB Guidelines

Introduction	8-1
Creating a Module to Convert to a UDFB	8-2
Software Declarations	8-4
UDFB Variables - Inputs and Outputs	8-4
Programming Notes	8-5
An Example of a UDFB and Module	8-6
Creating the UDFB	8-10
Using the UDFB	8-12
Type Checking	8-12
Editing UDFBs	8-13
Viewing the UDFB Module (LDO) from Another Module	8-15

CHAPTER 9-TASKS

Introduction	9-1
Programming	9-2
Creating a TASK	9-3
Interlocking Data when using Tasks	9-10
Setting up Semaphore Flags	9-10

CHAPTER 10-PiC and SERCOS

Introduction	10-1
Background Information	10-2
SERCOS Functions	10-2
Telegrams	10-3

CHAPTER 11- Stepper Axis Module

Introduction	11-1
Servo Setup	11-1
Notes on using Motion Library Functions and Variables with the Stepper	11-2

Appendix A - Module Filenames A-1

Appendix B -PiCPro Function Errors and Error Codes

General Errors	B-1
I/O Function Block Error Codes	B-1
STRING Function Errors	B-4

Appendix C - PiCServoPro Error Codes..... C-1

Appendix D - Function Input/Output Data Type Reference..... D-1

Appendix E - PiCServoPro Reference Card - Errors/Variables E-1

Appendix F - Stepper Reference Card..... F-1

Appendix G - Time Axis G-1

Appendix H - Command Line Switches..... H-1

Appendix I - Diagnostic LED Error Codes..... I-1

Appendix J - IBM ASCII Chart J-1

Application Note 1..... AN-1

Index i

Introduction to the Software Manual

The software manual covers information needed to program the PiC900 with the PiCServoPro or PiCPro software packages. The PiCServoPro software package is required for motion control applications. The PiCPro software package is required for all other PiC900 applications.

Table 1. Summary of Software Manual

Read	Title	For information on
Chapter 1	Getting Started with PiCPro for Windows	<ul style="list-style-type: none"> • Software installation procedures • Workstation and printer setup
Chapter 2	Introduction to PiCPro and Programming	<ul style="list-style-type: none"> • Programming with PiCPro
Chapter 3	Programming and Declarations	<ul style="list-style-type: none"> • Creating an application program • Entering software and hardware declarations
Chapter 4	System Integration	<ul style="list-style-type: none"> • Commands used to interact between the software and the PiC900
Chapter 5	Motion Control Concepts	<ul style="list-style-type: none"> • Motion control with PiCServoPro
Chapter 6	Servo Setup and Tuning	<ul style="list-style-type: none"> • Entering setup data for axes • Tuning axes
Chapter 7	Axis Profile Setup	<ul style="list-style-type: none"> • Creating ratio profiles for master/slave applications
Chapter 8	UDFBs	<ul style="list-style-type: none"> • Offers guidelines writing an LDO to convert to a UDFB and how to use the UDFB feature
Chapter 9	TASKs	<ul style="list-style-type: none"> • Offers guidelines writing an LDO to convert to a TASK and how to use the TASK feature
Chapter 10	PiC and SERCOS	<ul style="list-style-type: none"> • Offers guidelines when working with SERCOS and the PiC
Chapter 11	Stepper Axis Module	<ul style="list-style-type: none"> • Provides software information when working with the stepper axis module
Appendix A	Filenames	<ul style="list-style-type: none"> • Definitions of all filenames
Appendix B	PiCPro Function Errors and Error Codes	<ul style="list-style-type: none"> • Function errors and I/O function block error codes
Appendix C	PiCServoPro Error Codes	<ul style="list-style-type: none"> • Motion control errors
Appendix D	Data Types of Function Inputs/Outputs	<ul style="list-style-type: none"> • Alphanumeric list of inputs/outputs to functions/blocks, giving data type of each
Appendix E	PiCServoPro Reference Card	<ul style="list-style-type: none"> • PiCServoPro errors and variables
Appendix F	Stepper Reference Card	<ul style="list-style-type: none"> • Stepper motor control module (SMCM) commands, control words, status, and errors
Appendix G	Time Axis	<ul style="list-style-type: none"> • An explanation of the time axis
Appendix H	Command Line Switches	<ul style="list-style-type: none"> • List DOS switches that can be entered on the command line
Appendix I	Diagnostic LED Error Codes	<ul style="list-style-type: none"> • Describes the flashing LED error codes on the CPU module.
Appendix J	ASCII Codes	<ul style="list-style-type: none"> • A chart of the IBM ASCII codes
	Application Note	<ul style="list-style-type: none"> • Engineering note

There is also a Hardware Manual and a Function/Function Block Reference Guide.

NOTES

CHAPTER 1 Getting Started with PiCPro for Windows

Introduction to PiCPro for Windows

PiCPro for Windows is software packages available to program the programmable industrial computer (PiC900/PiC90).

- Project manager - allows you to manage an application through development and production phases and then maintain the application
- Control programmer (PiCPro) - an off-line ladder logic programming package used to create and edit application programs (also called *software modules*)
- Servo setup and tuning (SrvSetup) - an axes setup and tuning program
- Sercos setup (SRCSETUP) - a SERCOS setup program
- Axis profile setup (PiCPfl) - a program used in master/slave applications to create ratio profiles where the motion of a slave axis relative to the position of a master axis is defined

PiCPro may be used alone if the application does not require motion control.

This chapter provides information on how to get started using the software. It covers the following:

- installing PiCServoPro (or PiCPro) on the computer workstation
- using PiCServoPro (or PiCPro)
- setting up the workstation and its serial port
- configuring a printer to print out ladder diagrams and documentation
- using the project manager

The Computer Workstation

The PiCServoPro/PiCPro software runs on IBM PC compatible platforms. The requirements for a workstation are found in Table 1-1.

Table 1-1. Work station requirements

	Recommended
Computer	An 80386 or 80486 based IBM PC or 100% compatible
Memory	2 MB of RAM with at least 580 K of memory available to PiCPro*
Monitor	IBM VGA display adapter and monitor
Disk drives	Hard drive to install the software on, plus one floppy diskette drive for the installation disks. PiCServoPro needs at least 1.2 M of disk space. The floppy drive may be 3 1/2" (720K) or 5 1/4" (360 K).
Serial communications port to PiC900	RS232 port
MS DOS version	6.2

*It is recommended that you use DOS 6.2 facilities to load DOS "HIGH" between 640 K and 1 MB. Use the memory above 1 MB for a RAMDISK to store temporary files generated by PiCPro. Leave at least 256 K of memory above 1 MB for use by PiCPro. You can tell PiCPro to use the RAMDISK for temporary files by using the DOS SET command in your autoexec.bat file.

SET TMP = Drive:\Directory

Hardware Connections

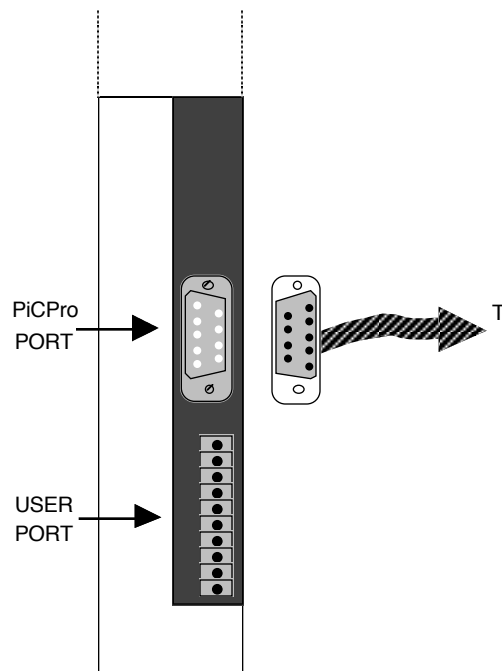
A cable is supplied with the PiCPro software disks, with one end labeled "PiC900" and the other "Computer". Connect the "Computer" end to the serial port you specified in the Computer dialog box.

IMPORTANT

Be careful to plug the "Computer" end of the cable into the workstation serial port, and the "PiC900" end into the PiCPro Port on the CPU or CSM/CPU module.

Figure 1-1 shows how the cable is connected to the PiCPro Port.

Figure 1-1. Hardware Connection to the PiCPro Port on the PiC900 CPU Module



The pin-out for the CPU serial port is given in the hardware manual. The pin-out for the workstation serial port should be in the computer manual.

PiCServoPro/PiCPro Installation

The PiCServoPro or PiCPro software is supplied on one disk.

Installing PiCServoPro or PiCPro is a matter of following prompts on the workstation screen. The PiC(Servo)Pro files are installed to and maintained in two directories. The installation procedure creates the two directories. The option to use default directory names is provided.

Installing PiCServoPro on the hard disk

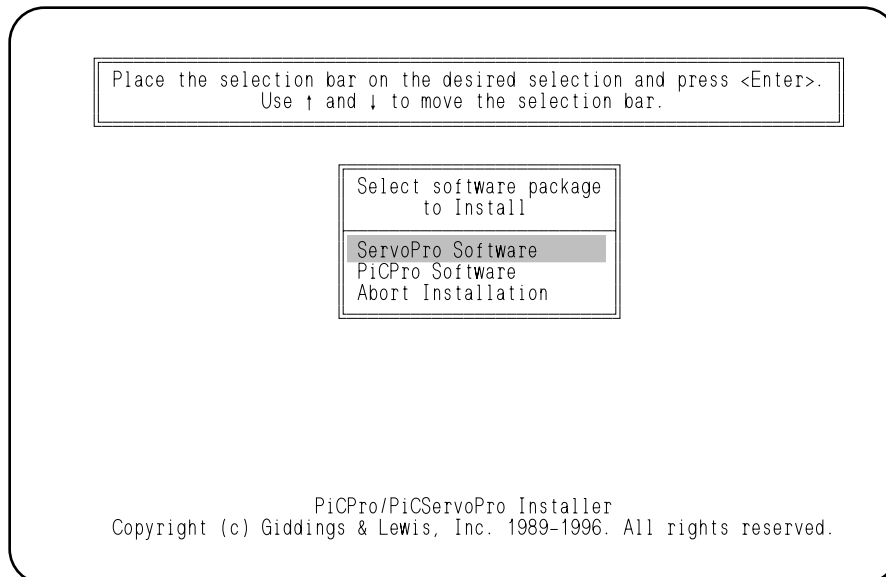
In the procedure below, type the words given in **bold type**. A symbol such as <Enter> or <F10> means that you should press that key.

Insert the Program Disk in Drive A:.

Type **a:install**. The screen in Figure 1-2 appears.

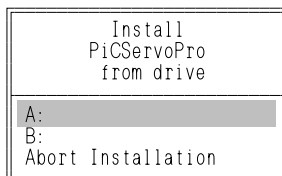
Use the arrow keys to highlight an option. Press <Enter>.

FIGURE 1 - 1. Select PiCServoPro or PiCPro Software



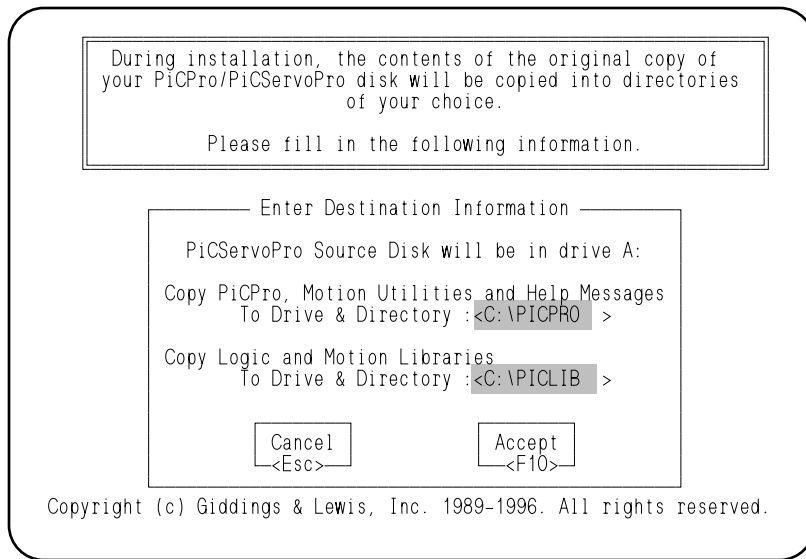
Most screens in the installation procedure give you an Abort option. If you want to stop installation, use the down-arrow key to highlight Abort and press <Enter>. The DOS prompt appears.

If you Abort after the directories are created, this message appears. Type any key to get the DOS prompt. The directories remain on disk, together with any files that have been installed in them. Use DOS commands to remove them.

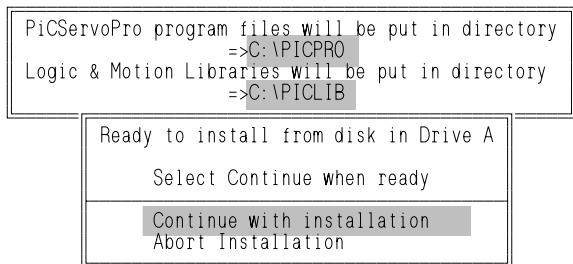


To continue installation, use the arrow keys to make a selection and press <ENTER>.

FIGURE 1 - 2. Destination Screen when Installing on a Hard Disk



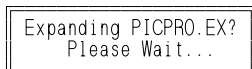
Press <F10> to accept the default directory names.



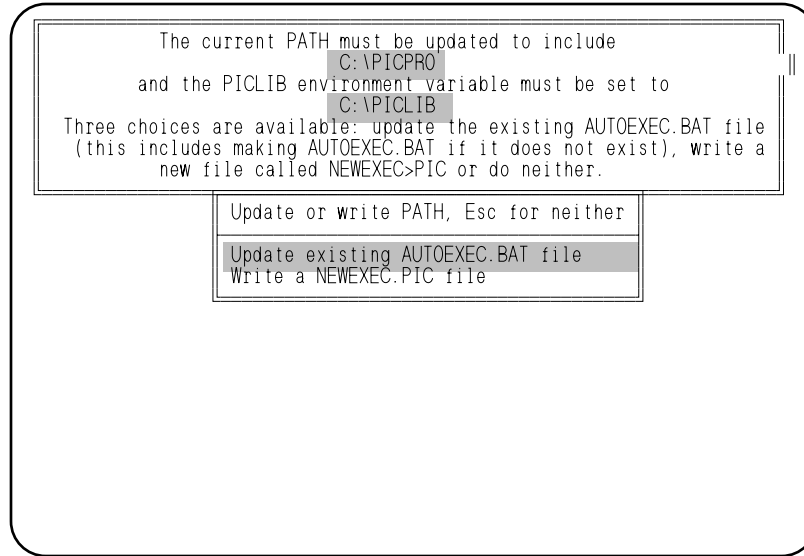
Check the directory names in the upper box. If they are not correct, select Abort and start over.

If the directory names are correct, press <Enter>.

The “Expanding” message box appears.

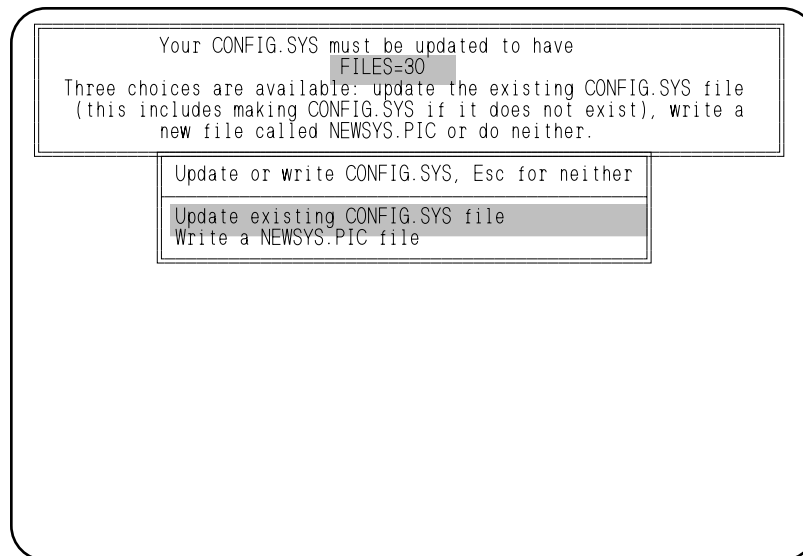


The following AUTOEXEC.BAT screen appears giving you the three choices



described.

The following CONFIG.SYS screen appears again offering three choices.



The installation software can automatically change the AUTOEXEC.BAT and CONFIG.SYS DOS files, or create new files to accommodate PiC(Servo)Pro.

IMPORTANT
<p>The AUTOEXEC.BAT file should include these two statements at the end of installation:</p> <p>PATH=C:\PiCPro (or the directory name you supply). SET PICLIB=C:\PiCLib (or the directory name you supply).</p> <p>The CONFIG.SYS file should contain the line:</p> <p>FILES=30</p>

The CONFIG.SYS file may need the following line also:

```
DEVICE=C:\DOS\ANSI.SYS
```

Install ANSI.SYS in the CONFIG.SYS file if panning across a graphed profile (created with PiCServoPro) causes the new screen to overwrite the old screen instead of erasing it. This is caused by older or incompatible graphics cards.

When all the files have been copied to the specified directories, a final screen appears.

<p>PiCServoPro Installation Complete.</p> <p>Don't forget to reboot to set PATH and FILES.</p>
--

Possible installation problems

<p>Incorrect diskette in drive A: Please put the correct diskette in the drive.</p>

If this message appears, insert the correct disk and press <Enter>.

Certain computer clones are not completely compatible with IBM XT or AT. If you get this message when the disk is correct, try installing PiCServoPro on a different computer. If another computer is not available, it can also be installed manually. Consult Giddings & Lewis.

<p>Not Enough Room on target drive for the utilities PiCPro utilities will not be installed! Please see your manual for instructions on installing a Subset of the utilities.</p> <p style="text-align: center;">Press any key to continue</p>
--

If this message appears, the installation procedure has already stopped; pressing a key returns you to the DOS prompt. Use the **chkdsk** command to see if hidden files might be reducing free disk space below the minimum required size.

If you cannot make room for PiCServoPro, you may need to install its development tools individually.

Using PiCServoPro/PiCPro

Create another directory on your hard disk in which to store the programs you create. If you create many application programs, you might need more than one applications directory with names that relate to their contents.

Change to the drive and directory where the application programs are stored before you load PiCServoPro or PiCPro, as shown in Table 1-3.

NOTE ON UPGRADES

When you receive a new version of PiCServoPro/PiCPro software from Giddings & Lewis, use the /u switch on the command line when starting PiCPro for the first time.

C:/picpro /u

This will cause *all* the UDFBs you have in your system to be updated. It is necessary to do this to ensure that your system will function properly.

It is important that the /u switch be used on *all directories containing UDFBs*, either through the PICLIB environment variable or by default.

Whenever you receive a new version of PiCServoPro/PiCPro software from Giddings & Lewis, also complete these tasks to ensure that your system will function properly.

PiCPro scans the function libraries and, if it finds any UDFBs that call other functions created at a later date, it will ask if you want PiCPro to recompile these UDFBs automatically. Answer **Yes**.

Run the ServoSetup program and remake your servo setup function. Download your program to the PiC.

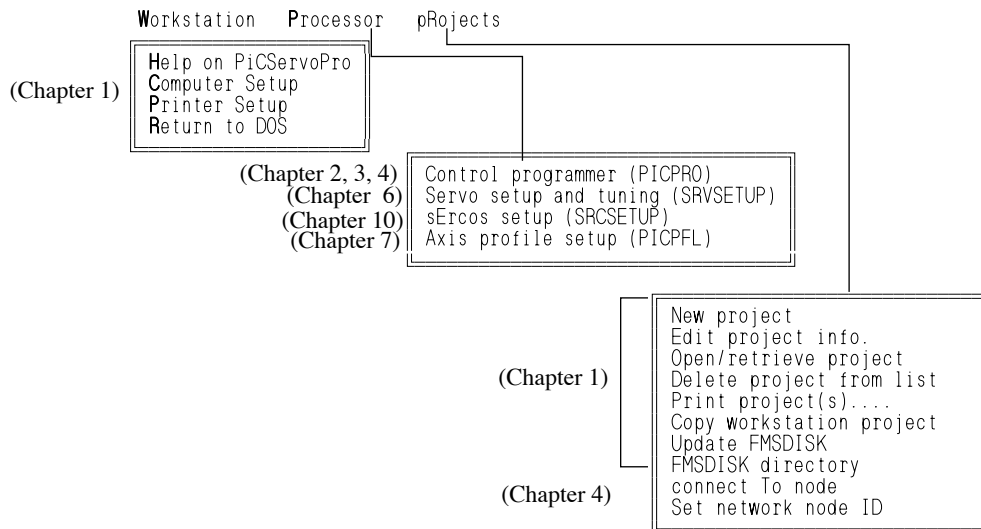
Table 1-1. Starting up PiCServoPro or a Development Tool

To run PiCServoPro	To run PiCPro alone
At the DOS prompt, change to the directory where your application programs are stored, and then type picservo <Enter>.	At the DOS prompt, change to the directory where your application programs are stored, and then type picpro <Enter>.

FIGURE 1 - 3. PiCServoPro Initial Screen



The following menus appear under the three items on the menu bar. The chapter numbers on the left tell you where you can find more information on them.



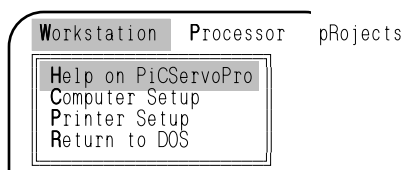
Menu Bars, Menus and Pop-Ups

The top line of this screen is called the *menu bar*. Every PiCServoPro or development tool screen has a menu bar, which contains a row of menu items. If you highlight a menu name and press <Enter> a menu drops down under the name, with its top item highlighted. Use the down-arrow key to highlight any other item, then press <Enter> to select the highlighted item. Depending on the item, one of the following appears:

- An entirely new screen with a different menu bar.
- Another menu whose name is the item you selected.
- A pop-up selection box with several options to select from.
- A pop-up dialog box with blank entry fields to fill in. If you highlight an entry field and press <F8>, you usually get a *selection list* of all available choices. Highlight a choice and press <Enter> to put it in the entry field. Or type data in the entry field.
- A pop-up message box or a message in the lower left corner of the screen that tells you what task is going on. If the message merely confirms what you asked for, it may come and go. Otherwise it remains on screen until you press a key to erase it.

Hot Keys

One letter in every item on the menu bar, and in each submenu, dialog box or selection list, is in **bold** style. This letter, always capitalized and usually the first one, is called a *hot key*. If you press a hot key, the corresponding option is selected at once.



For instance, in the screen in Figure 1-4, press **W**. The Workstation menu drops down, with the top item, Help on PiCServoPro, highlighted.

If you then press either **H** or <Enter>, the PiCServoPro Help message appears.

<F9> Help key

PiCServoPro has a *context-sensitive help system*. That is, each help screen is different depending on where the highlighting is. Call for HELP at any time by pressing the <F9> key. As a general rule, get as close to a problem as possible before you press <F9>.

When the PiCServoPro initial screen appears, the menu name Workstation is highlighted. If you press <F9> the Help message briefly explains the items in that menu. Press **W** so the Workstation menu drops down, and then use the down-arrow key to move the highlight down the menu. If you press <F9> for each item, you can see that the Help message applies only to the highlighted item.

General-purpose keys

- is used in programming to delete a highlighted network or an entry in a table. When you press you always get a pop-up that asks you to confirm that you want to delete the item.
- <Enter> Accepts a choice on a list or a menu or in a dialog box. In Figure 1-1 at the start of the installation procedure, you used <Enter> to choose either PiCServoPro or PiCPro.
- <Esc> Cancels something. If you called up a dialog box, menu or list by mistake and just want to erase it from the screen, press this key. In Figure 1-2, <Esc> would cancel anything you typed in the blanks and restore the default directory names.
- <F8> is the List key. If you highlight an entry field on a dialog box, this key will usually bring up a selection list of all available items that could be used in that entry field. Highlight an item on the list and press <Enter>. It is copied into the entry field.
- <F9> is the Help key. Press this key at any time to call up a help screen. The message that is displayed is relative to the function you are performing.
- <F10> Accepts a number of choices at once. It is used for a dialog box with a number of entry fields or a table that must be filled in. You must use <F10> to Accept an edited network, since <Enter> Accepts only elements within the network. In Figure 1-2, <F10> confirmed the names of the directories to be used in the installation procedure.
- <Ins> is used in programming to make room for a new network or a new entry in a table. The highlighted item is moved down one space so the new material can be entered above it.
- ←, ↑, ↓, → The arrow keys allow you to move around in a menu, list, or network until the item you want to work with is highlighted.

Certain PiCServoPro screens use such key combinations as <Alt>E or <Alt>M for special purposes. These uses are indicated on screen or in a Help message.

Setting Up the Workstation Software

The workstation and the PiC900 system must be able to communicate with each other before you can send a PiCPro application program to the system. Communication requires both software and hardware compatibility.

Workstation menu

PiCServoPro/PiCPro software requires certain information about the workstation so it can communicate with the monitor screen and the PiC900 system. The Workstation - Computer setup menu item is identical in all development tools. Thus if PiCPro has the correct parameters, Servo Setup does also.

```
Computer setup
Color Display: No
Remove Snow: No
Display lines
Press <F8> for choices: 25
Control connected to Com [1 or 2]: 1
Communication baud rate
Press <F8> for choices: 57600
Changes will be implemented with
the next start of this program
Cancel <Esc>
Accept <F10>
```

AA1106-1692

Press **WC** to bring up the Computer Setup dialog box. If all the default choices are correct, press **<F10>** to Accept them and return to the Workstation menu. **<Esc>** simply erases the dialog box and leaves the default options unchanged.

<Enter> or the down-arrow key highlights each option in turn:

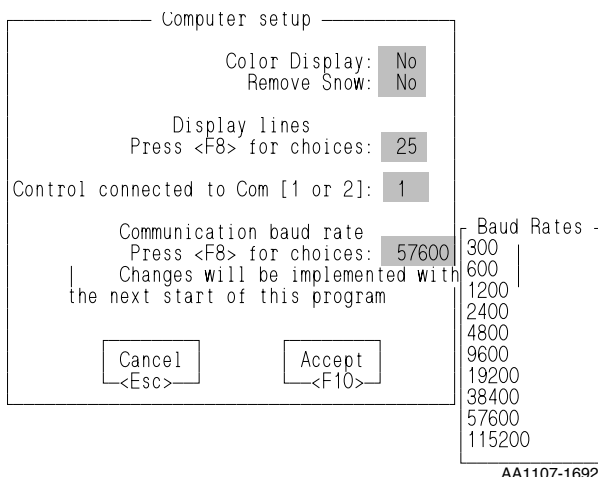
Color Display: If the workstation display screen is black-and-white, press **N** to change the default.

Remove Snow: IF you have a CGA adapter board and IF the screen flickers during updates, press **Y**. Use this option only if you must, since it slows screen updates.

Display lines: The default for the number of lines displayed on the screen is 25. An EGA monitor can be set to 43 and a VGA monitor can be set to 50.

Press **<F8>** to bring up the list of these three choices.

Control connected to: You must specify which computer port will be used to communicate with the PiC900 system.



Communication baud rate: The default baud rate for communicating between PiCPro and the PiC900 through the communications port is 57600.

When you want to operate your system at a baud rate other than 57600, you can change the baud rate here. Press <F8> to bring up a list of available baud rates as shown on the left.

Move the cursor to the rate you want and press <F10> to accept the new rate.

You can repeat this procedure anytime you want to change to a different rate.

NOTE: Not all computers can perform at the 115200 baud rate.

If you press <Esc> any changes you made in the computer dialog box are ignored.

NOTE

If you made any changes in the Computer dialog box, restart the development tool so the software has the correct settings.

- Exit to DOS.
- At the DOS prompt type the name of the development tool.

Temporary baud rate change

When you need to operate at a different baud rate for a short time, you can change the baud rate by typing in a new rate on the command line. Enter a /B and the desired rate after the PiCPro command. The example below will enter a baud rate of 9600.

PiCPro /B9600

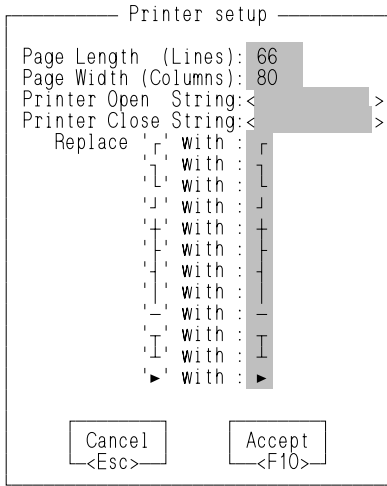
If any number other than the baud rates listed in the Computer dialog box is entered on the command line, the PiC900 will not lock on. To see if communications has been established, check the lower right hand corner of the screen to the left of the time. An up arrow indicates PiCPro is still looking for the PiC900. A double-headed arrow indicates that communications between PiCPro and the PiC900 is established.

This temporary baud rate will remain in effect until you restart PiCPro.

NOTE: Lower rates will slow down communication tasks such as downloading, monitoring, etc.

Configuring the Printer

Before you change anything in this dialog box, use the self-test on the printer to make sure it works properly and use DOS commands to check that the computer workstation can communicate with the printer.



AA132-1190

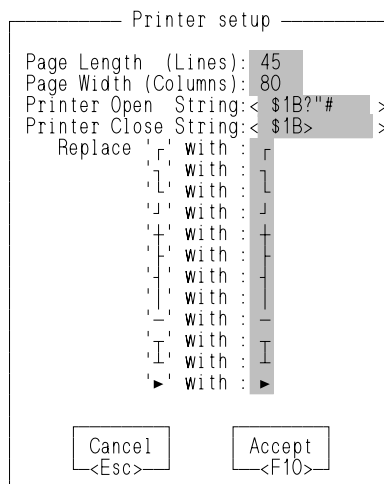
Press WP (Workstation - Printer) to bring up the dialog box.

Page Length/Width: defines the size of the printed page. The default setting assumes the printer uses paper 11" long (at 6 lines to the inch) and 8.5" wide (at 10 characters to the inch with 0.25" borders). If the printer has a wide platen, you must change the Page Width, since PiCPro measures the page in columns rather than inches or centimeters. For instance it can print 132 columns on paper 14" wide.

Printer Open/Close String: allows you to send special characters to the printer before and after the material to be printed is sent. This gives you the option of using various modes that may be available with your printer. Consult your printer manual.

If your printer requires an ESCAPE in the open or close string, use the hex code (\$1B) shown in the example below. ASCII symbols can be used in combination with the hex symbols.

FIGURE 1 - 4. Example of Printer Configuration Open and Close Strings.



Printing Extended ASCII Characters

The application program you create on the workstation screen uses the normal computer keys, plus twelve other text symbols supplied by PiCPro.

PiCPro uses the IBM extended ASCII character set to draw wires, functions, and function blocks.

If your printer is IBM compatible, your hard copy should be correct. If it doesn't look like a ladder diagram, try these suggestions in sequence.

1. Check your printer manual to see if you can set DIP switches or a dial pointer for the "IBM extended ASCII" character set.
2. The printer may use software rather than hardware to specify its character set. If so, its manual gives you a Printer Open String that selects the IBM extended ASCII character set while PiCPro is sending to the printer, and a Printer Close String that returns it to normal for other uses. Insert these paired codes in the blanks for the third and fourth options in the dialog box. If you use Condensed Mode also, the printer manual has another pair of strings that opens and closes both options together.
3. If you have a printer that doesn't print IBM extended ASCII, substitute
 - a "+" sign for the all the symbols marking corners,
 - a "!" sign for the vertical bar (fifth from the bottom),
 - a "-" sign for the horizontal bar (fourth from the bottom),
 - a ">" sign for the triangle at the bottom of the list.

These are regular punctuation marks, so any printer can recognize them.

Project Manager

Introduction

The project manager is a tool to help you manage your applications. It creates a description of your project defining all related files and their locations on disk. You can then edit, delete, or print this description as required.

Once the project files have been defined, you can work on the development of your application using the tools of PiCPro, servo setup, SERCOS setup, and/or the profile editor. A compressed file containing all the files identified in the project description can be created. This file can be stored on the FMSDISK. Any UDFB source and library files required by the project will be automatically included when the FMSDISK is updated.

The project manager can play a role in simplifying your control maintenance. You can maintain the PiC control by using a PC with only the PiC programming tools installed. All the files required for your application can be stored in the PiC on the FMSDISK. When you open a project on your PC from the FMSDISK, the compressed file is loaded onto the PC. The name of the project is automatically inserted in the project list. The compressed file is exploded into the subdirectory structure and filenames as described in the project description. You can then use the software tools to maintain the project. When finished, the project manager automatically scans all files and determines whether the file resident on the PiC FMSDISK requires updating and prompts you accordingly.

A summary of the capabilities of the project manager are listed below.

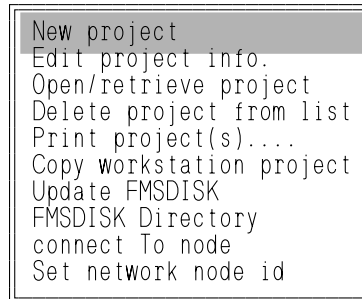
- Allows you to control the development of a project in a development, production, and archive version.
- Allows you to work with multiple versions of PiCPro. It eliminates the need to change the path in your AUTOEXEC.BAT file and to enter a SET PICLIB = statement everytime you want to switch to a different version of PiCPro.
- Simplifies creating a single disk containing all the files required for an application.
- Provides the tools needed to store a single compressed file containing all the files required for an application on the FMSDISK in the PiC.
- Allows you to update the FMSDISK on the PiC without stopping the scan when working in version 9.0 or higher.

Using Project Manager

Begin the project manager by running PiCServoPro version 9.0 or higher.

1. From the main menu, press **R** (p**R**ojects). This brings up the menu shown below. Press **N** (New) to define a new project description.

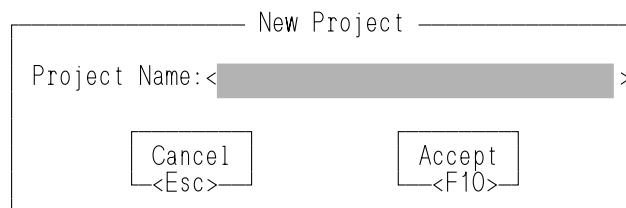
Workstation Programs pProjects



When you create a new project description, it is put in the project database which is a file called PICPRO.PRJ in your PICPRO directory.

The remaining commands allow you to edit, open, delete, print, copy, or update your FMSDISK with a copy of the project description, view the FMSDISK directory, connect to a different control on the network, and set the network node ID.

2. When you are defining a new project description, the box below appears. Enter the name of the new project (up to 80 characters). Press <F10> to accept. This will create a new project in the project database.



3. If the project is new, you will specify the directories and paths in the box below.

In *PiCPro Path*, enter the path including the drive to where the version of PiCPro you want to use with this project is located.

In *PiCLib Path*, you typically will include multiple paths separated by semicolons; one to the standard libraries in PiCPro and others to UDFB libraries which are stored in a subdirectory other than your main project directory.

In *Main .LDO Path*, you set the single path including the drive to where the main module for this project is located. This subdirectory is also where you must store all of the servo setup, SERCOS setup, profiles, and project specific UDFBs.

In *UDFB.LDO Path*, you set the path(s), separated by semicolons, to the

source code for all the UDFBs used in this project including those stored in the main project directory.

Press <F10> to accept the project data.

```
Enter Project Data
Project: WRAPPER MACH
PiCPro Path: <C:\PICPRO90; >
PICLIB Path: <C:\PICPRO90;C:\PICPRO90\ASFB; >
Main .LDO Path: <C:\PICPRO90\WRAPPER >
UDFB .LDO Path: <C:\PICPRO90\ASFB; >
Cancel Accept
<Esc> <F10>
```

4. The box below appears. Enter the appropriate files for the project. If there are multiple setup files, separate each with a semicolon.

At *File with list of Other Files*, specify the list as @filename.ext where filename.ext is the name of a file that contains the list of additional files to include.

At *Compressed File*, enter the name for a file that the project manager will create. Typically, this would be the same name as your main LDO. This file will contain all the files you listed in a compressed form to send to the FMSDISK using the project manager's Update FMSDISK command.

Press <F10> to accept the project data.

```
Enter Project Data
Project: WRAPPER MACH
Main Ladder File [.LDO]: WRAPPER
Servo Setup File(s) [.SRV]: <W1_SERVO >
Servo Setup Library(s) [.LIB]: <ZSERVO >
Sercos Setup File(s) [.SRC]: < >
Sercos Setup Library(s) [.LIB]: < >
Profile Setup File(s) [.PRO]: < >
Profile Setup Library(s) [.LIB]: < >
File with list of Other Files:
Compressed File [.G&L]: APP_1
Cancel Accept
<Esc> <F10>
```

5. Now you have all the information describing this project in the project manager database. An example is shown below. Notice the box on the right. It is the list of any files you entered in Step 5. By placing the cursor on the appro-

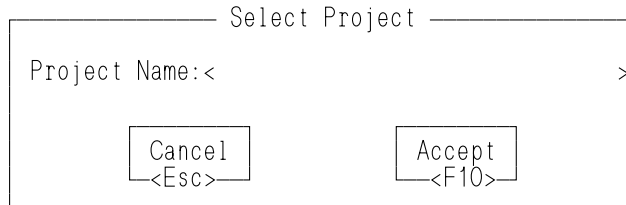
ropriate file and pressing <Enter>, you open the file using the PiCPro version you specified in the PiCPro Path entry of your description. In this example, version 9.0 is being used.

NOTE: Whenever you want to exit this page, press <Esc> to return to the PiCServoPro menu.

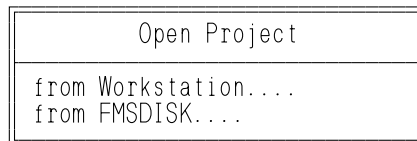
```
Name :WRAPPER
PiCPro Path      :C:\PICPRO90
PICLIB Env.Variable :C:\PICPRO90;C:\PICPRO90\ASFB
Main .LDO Path   :C:\PICPRO90\WRAPPER
UDFB .LDO Path   :C:\PICPRO90\ASFB;PICPRO90\WRAPPER
Project Details:_____
Main .LDO Name   :APP1.LDO
Servo .SRV Name(s) :W1_SERVO.SRV
Servo .LIB Name (s) :ZSERVO.LIB
Sercos .SRC Name (s) :....Unspecified....
Sercos .LIB Name (s) :....Unspecified....
Profile .PRO Name (s) :....Unspecified....
Profile .LIB Name (s) :....Unspecified....
Other File(s)    :....Unspecified....
```

Select a File or <Esc>
WRAPPER.LDO
W1_SERVO.SRV

- If you want to edit an existing project description, press R (pRojects), E (Edit). The following box appears. Press <F8> to bring up a list of projects in the project database.



- If you are opening an existing project, it can either come from the workstation or the PiC's FMSDISK.



When opening an existing project from the workstation, either type in the name of the project or press <F8> to bring up a list of projects. Move the cursor to the project you want to open and press <Enter>.

8. If you want to print the description of a single project or all projects in the database, press R (pRojects), P (Print). The box below appears. Make your selection and press <Enter>.

Print Project(s)
Single project
All projects

If you print a single project description, the select project box appears and you choose the project from the <F8> list or enter the name yourself. Below is an example of a single file print out.

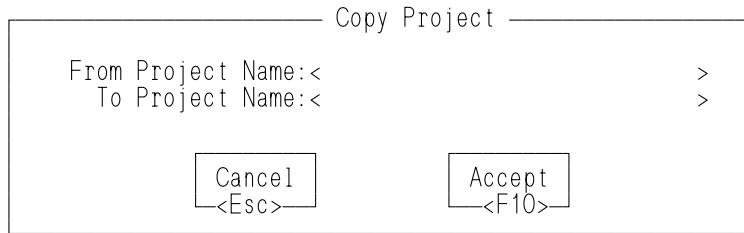
```
Project Project 1: WRAPPER
Name :WRAPPER Ladder
PiCPro Path      :C:\PICPRO90
PICLIB Var       :C:\PICPRO90;C:\PICPRO90\ASFB
Main .LDO Path   :C:\PICPRO90\WRAPPER
UDFB .LDO Path   :C:\PICPRO90\ASFB;C:\PICPRO90\WRAPPER
Main .LDO Name    :WRAPPER.LDO
Servo .SRV Name  :W1_SERVO.SRV
Servo .LIB Name   :ZSERVO.LIB
Sercos .SRC Name  :...Unspecified...
Sercos .LIB Name  :...Unspecified...
Profile .PRO Name :...Unspecified...
Profile .LIB Name :...Unspecified...
Other File(s)    :...Unspecified...
Compress into    :APP1.G&L
```

If you choose to print all the project descriptions, the following Print Setup box appears. The <F8> list contains the .LST file and you can select from it or enter the filename.

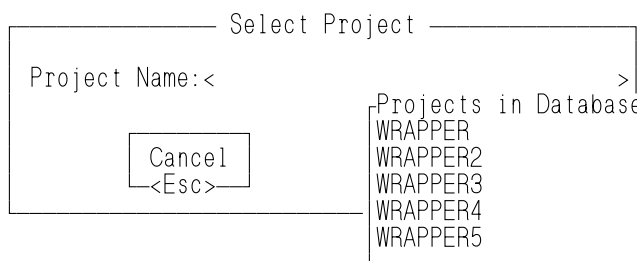
Print Setup	
List Filename :	<PICPROJ.LST >
Cancel	Accept
<Esc>	<F10>

9. The Copy command allows you to copy an entire project. This is a way you can move your development project to a production version.

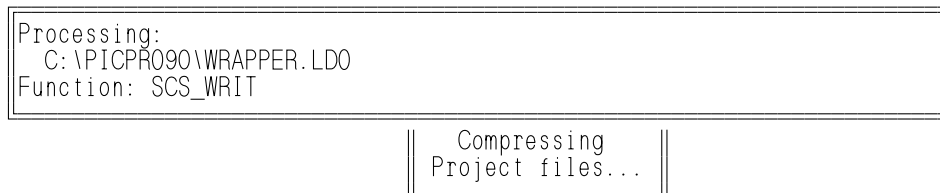
Enter the name of the project you want to copy from and the name of the new project location. Press <F10>.



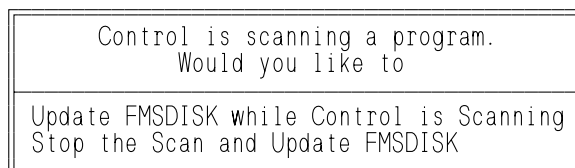
10. The Update FMSDISK command allows you to update the FMSDISK on the PiC with a compressed file containing the current project. When using PiCServoPro version 9.0 or later, the FMSDISK can be updated without stopping the scan. Select the project from the <F8> list or type it in. Press <F10> to accept.



The message below appears as the project is being prepared to send to the FMSDISK.



11. When the processing is complete, the following screen appears. You can chose to update the FMSDISK with or without the scan running on the PiC. Move the cursor to make your choice and press <Enter>.



After the FMSDISK is updated, the following message appears.

```
FMSDISK: Updated.  
Control is Scanning Program.  
Press any key to continue
```

If you are using an earlier version of PiCPro than version 9.0, the scan will have to be stopped in order to update the FMSDISK. The following warnings appear in succession.

```
WARNING: The Update Disk command will Stop Scan,  
erase the FLASH Disk and write new information.  
  
Are you SURE you want to Continue?  
  
Yes  
No
```

```
Ready to Stop Scan on Control & Update FLASH Disk!  
Press any key to continue
```

The following message appears as the FMSDISK is being updated.

```
Formatting & Updating FMSDISK:  
Please Wait ....
```

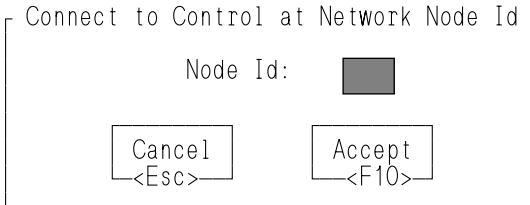
When the update is complete, the following message reminds you that the scan on the PiC is not running.

```
FMSDISK: Updated  
Control is not Scanning Program  
Press any key to continue
```

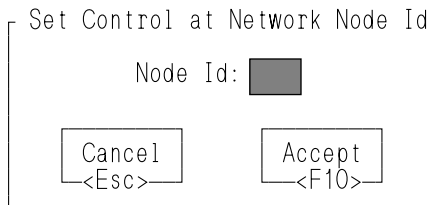
12. The FMSDISK Directory command allows you to see the file currently in the PiC FMSDISK.

```
Files/Dirs in  
CONTROL:FMSDISK:\*.* <8245248 formatted bytes free>  
WRAPPER.G&L 358 97-03-211:00a
```


13. The Connect to node command allows you to connect to a different PiC on the network.



14. The Set network node id command allows you the set the node id for this node.



Subdirectory Structures

When you want to create a production version of the development application you are working on, use the copy command. The table below illustrates the subdirectory structure.

Development Version of the Project	Production Version of the Project
<p>Project 1: WRAPPER Name : WRAPPER Development Version PiCPro Path : C:\PICPRO90¹ PICLIB Env. Variable: C:\PICPRO90²;C:\PICPRO90\ASFB³ Main .LDO Path : C:\PICPRO90\WRAPPER\DEV⁴ UDFB .LDO Path : C:\PICPRO90\WRAP- PER\DEV⁵;C:\PICPRO90\ASFB⁶ Main .LDO Name : WRAPPER.LDO Servo .SRV Name(s) : W1_SERVO.SRV Servo .LIB Name (s) : ZSERVO.LIB Sercos .SRC Name (s) :Unspecified.... Sercos .LIB Name (s) :Unspecified.... Profile .PRO Name (s) :Unspecified.... Profile .LIB Name (s) :Unspecified.... Other File(s) :Unspecified.... Compress into : WRAPPER.G&L</p>	<p>Project 1: WRAPPER Name : WRAPPER Production Copy PiCPro Path : C:\PICPRO90¹ PICLIB Env. Variable: C:\PICPRO90²;C:\PICPRO90\WRAP- PER\PROD\ASFB³ Main .LDO Path : C:\PICPRO90\WRAPPER\PROD⁴ UDFB .LDO Path: C:\PICPRO90\WRAPPER\PROD⁵;C:\PICPRO90\WRAP- PER\PROD\ASFB⁶ Main .LDO Name : WRAPPER.LDO Servo .SRV Name(s) : W1_SERVO.SRV Servo .LIB Name (s) : ZSERVO.LIB Sercos .SRC Name (s) :Unspecified.... Sercos .LIB Name (s) :Unspecified.... Profile .PRO Name (s) :Unspecified.... Profile .LIB Name (s) :Unspecified.... Other File(s) :Unspecified.... Compress into : WRAPPER.G&L</p>
<p>NOTE: During development, all ASFBs and their associated libraries are stored in a subdirectory: C:\PICPRO90\ASFB</p>	<p>NOTE: The process of copying the project will create a copy of all required ASFBs and their associated libraries in the main project subdirectory.</p>
<p>Footnotes are for subdirectories where the:</p> <ol style="list-style-type: none"> ¹version of PiC software tools (PiCPro 9.0 in this example) is stored. ²standard PiCPro library files are stored. ³standard ASFB library files are stored. ⁴application files (main .LDO, .SRV, etc.) are stored. ⁵project specific UDFB source files are stored. ⁶standard ASFB source files used by this project are stored. 	

CHAPTER 2 Introduction to PiCPro and Programming

Introduction

PiCPro is the tool you use to create, maintain, and monitor execution of a PiC900 software program. PiCPro is a menu driven system. All functions are performed by selecting menu items.

The software program created with PiCPro is called a *module*. A module is a ladder logic based program which incorporates higher level commands.

This chapter introduces PiCPro and the components of a module, showing how to use PiCPro to create a module.

Prerequisites for PiCPro Programming

A prerequisite to creating a module is to have a good understanding of ladder diagram logic. This manual shows how elements of ladder diagrams are entered with the PiCPro language, but it assumes that you are proficient with ladder programming.

Components of a Module

A module consists of three components - networks, a software declarations table, and one hardware declarations table for every rack (master and expansion).

Networks are the executable portion of a module. They contain ladder logic and higher level commands.

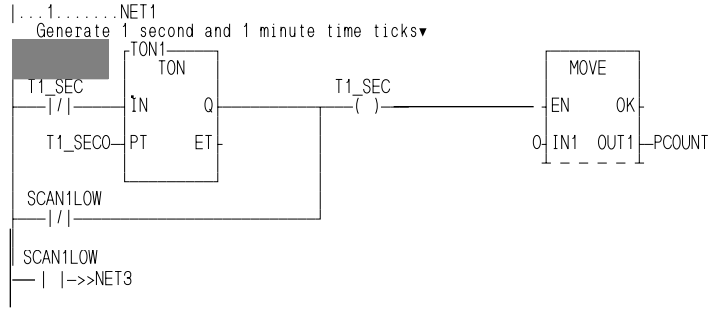
The *software declarations table* contains information about the data and about certain operations performed in the networks.

A *hardware declarations table* contains the locations of all hardware modules and input/output points.

Networks

PiCPro numbers networks consecutively. Each network is executed sequentially unless a command forces execution to be out of numerical sequence. Within a network as shown in Figure 2-1, the ladder logic is executed left to right, top to bottom.

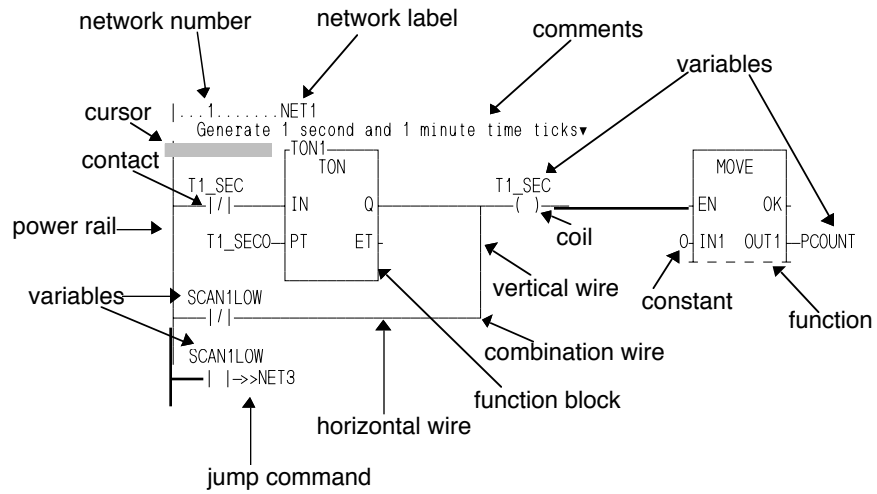
FIGURE 2 - 1. An Example Network



AA201-2790

Figure 2-2 labels the elements in a network. Descriptions of these elements are found in the table that follows.

FIGURE 2 - 2. Elements in a Network



AA203-2790

Elements of Networks

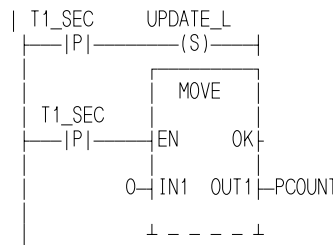
	Description
comments	Optional documentation for each network.
contacts, control relays, coils	Represent hardwired inputs and outputs and/or internal logic.
functions	Perform operations such as arithmetic or motion control.
function blocks	Perform operations that must retain data for a period of time, such as timing and counting operations.
jump command	Causes execution of the module to jump or move to a specified network.
label	Optional name assigned to a network, required for the jump command.
variables structures and arrays	Elements whose values can change. They are used as inputs to and outputs from functions and function blocks, and to represent the states of contacts, control relays, and coils. Group of variables that has a common link.

Network Size

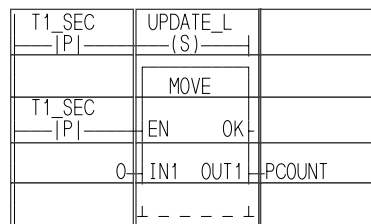
A network is an array or matrix that is comprised of rectangles, called *cells*. A cell is about the size of the cursor. Each element in a network occupies at least one cell. Some elements, such as functions, occupy several cells.

PiCPro accepts a network whose cells form a rectangle with an area less than or equal to 255. The area of a network is the product of its width times its length, where the width is the widest row and the length is the longest column.

Elements



Elements in cells



In the example above the UPDATE_L coil occupies one cell. Also the T1_SEC contacts and the PCOUNT variable occupy one cell each. The MOVE function occupies four cells. The entire segment of logic occupies fifteen cells as shown on the right.

Software Declarations Table

The data in the software declarations table tells PiCPro about the variables and function blocks in the networks and how much memory to reserve for each one.

FIGURE 2 - 3. Example Software Declarations Table

Name	Type	I/O Pt.	Init. Val.	Long Name
SW1	BOOL	I03.01		
SW2	BOOL	I03.02		
COIL1	BOOL	O07.01		
COIL2	BOOL	O07.02		
DURATION	TIME		1000ms	
PROMPT1	STRING(6)		READY!	
PROMPT2	STRING(5)		DONE!	
TIMER1	<fb>TON			

Elements of Software Declarations Table

	Description
Name	Unique name (maximum of eight characters) assigned to every variable and function block.
Type	Defines the type of data or function block and enables PiCPro to reserve memory for each element.
I/O Pt.	For a variable representing a physical input or output, tells PiCPro where the hardware module is located and which channel is being used.
Init. Val.	Assigns an initial value to a variable.

Hardware declarations table

The data in the hardware declarations tables tells PiCPro the configuration of the hardware modules in each rack. A module has a hardware declarations table for the master rack and a hardware declarations table for every expansion rack.

FIGURE 2 - 4. Example hardware declarations tables.

Master Rack

Expansion I/O Rack

Master Rack Hardware Declarations	Expansion Rack 1 Hardware Declarations
<pre> Press <F8> for hardware list, then select desired type and press <Enter>. Slot 1: CSM Slot 2: PiC900, 64K Slot 3: Empty Slot 4: Empty Slot 5: Empty Slot 6: Empty Slot 7: Empty Slot 8: Empty Slot 9: Empty Slot 10: Empty Slot 11: Empty Slot 12: Empty Slot 13: Empty Expansion I/O Type: None </pre>	<pre> Press <F8> for hardware list, then select desired type and press <Enter>. Slot 1: CSM Slot 2: I/O DRIVER Slot 3: Empty Slot 4: Empty Slot 5: Empty Slot 6: Empty Slot 7: Empty Slot 8: Empty Slot 9: Empty Slot 10: Empty Slot 11: Empty Slot 12: Empty Slot 13: Empty </pre>

Expansion Block I/O

Define Block I/O Hardware configuration			
01: Empty	21: Empty	41: Empty	61: Empty
02: Empty	22: Empty	42: Empty	62: Empty
03: Empty	23: Empty	43: Empty	63: Empty
04: Empty	24: Empty	44: Empty	64: Empty
05: Empty	25: Empty	45: Empty	65: Empty
06: Empty	26: Empty	46: Empty	66: Empty
07: Empty	27: Empty	47: Empty	67: Empty
08: Empty	28: Empty	48: Empty	68: Empty
09: Empty	29: Empty	49: Empty	69: Empty
10: Empty	30: Empty	50: Empty	70: Empty
11: Empty	31: Empty	51: Empty	71: Empty
12: Empty	32: Empty	52: Empty	72: Empty
13: Empty	33: Empty	53: Empty	73: Empty
14: Empty	34: Empty	54: Empty	74: Empty
15: Empty	35: Empty	55: Empty	75: Empty
16: Empty	36: Empty	56: Empty	76: Empty
17: Empty	37: Empty	57: Empty	77: Empty
18: Empty	38: Empty	58: Empty	
19: Empty	39: Empty	59: Empty	
20: Empty	40: Empty	60: Empty	
			<div style="display: inline-block; border: 1px solid black; padding: 2px; margin-right: 20px;">Cancel <Esc></div> <div style="display: inline-block; border: 1px solid black; padding: 2px;">Accept <F10></div>

Hardware declarations table

	Description
Slot 1	Reserved for the Central Services Module
Slot 2	Reserved for the CPU module (master rack), or the I/O driver module (expansion racks) Select the control type and then the CPU model type from the F8 list.
Slots 3 - 13	Defines the type and location of the I/O modules Select the module type and then the correct module from the F8 list.
Expansion I/O Type	Defines the type of expansion connected to the master rack.

There are three levels of menus in PiCPro. At each level various submenus exist. All functions performed in PiCPro are performed by selecting menu items. To create or maintain the components of a module some of the submenus at each of the three levels must be accessed. The submenus of the three menu levels are shown below. Only one submenu at a time can be displayed on the workstation screen.

Main menu

The Main menu is the first screen to appear in PiCPro (after the introductory sign-on screen).

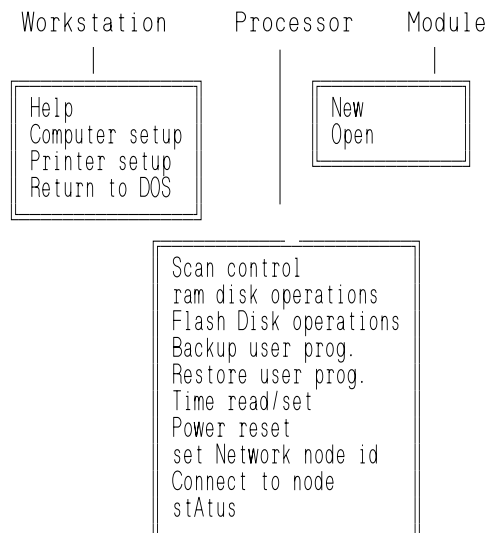
The Workstation option was presented earlier. It was shown that items under this selection are used to call up the Help system, set up the computer and printer, and to exit PiCPro. The Workstation option of the Main menu is identical to the Workstation option of the Module menu.

The Processor option has items that relate to the operations of the PiC900 such as scan control and power reset. The Processor option of the Main menu is identical to the Processor option of the Module menu.

The Module option is where a new or existing module name is entered by accessing New or Open respectively. Once a new name is entered, you can begin to build the networks for the module. Once an existing module is opened, you can edit, download, monitor, etc. that module.

NOTE: The New and Open items are available on the Module menu level also (see next section).

FIGURE 2 - 5. Main Menu

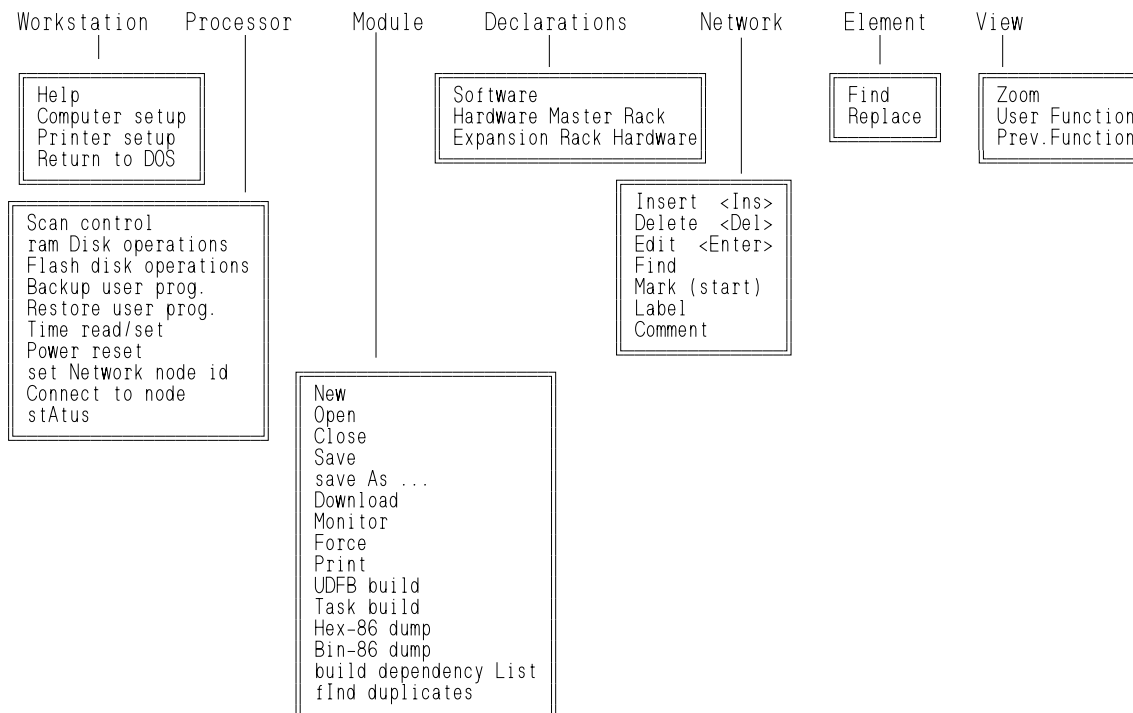


Module Menu

The Module menu is displayed after a module name is entered from the Main menu, or when the third level Network menu is exited. From this level the hardware declarations are made or changed. The software declarations can be made or changed from this level or from the Network menu level.

Other functions performed at this level include: save, close, download, print, and monitor execution of a module. Also available on the Module menu are editing commands such as search for an element, add a network label, add comments to a network, insert a network, and zoom the view of a network.

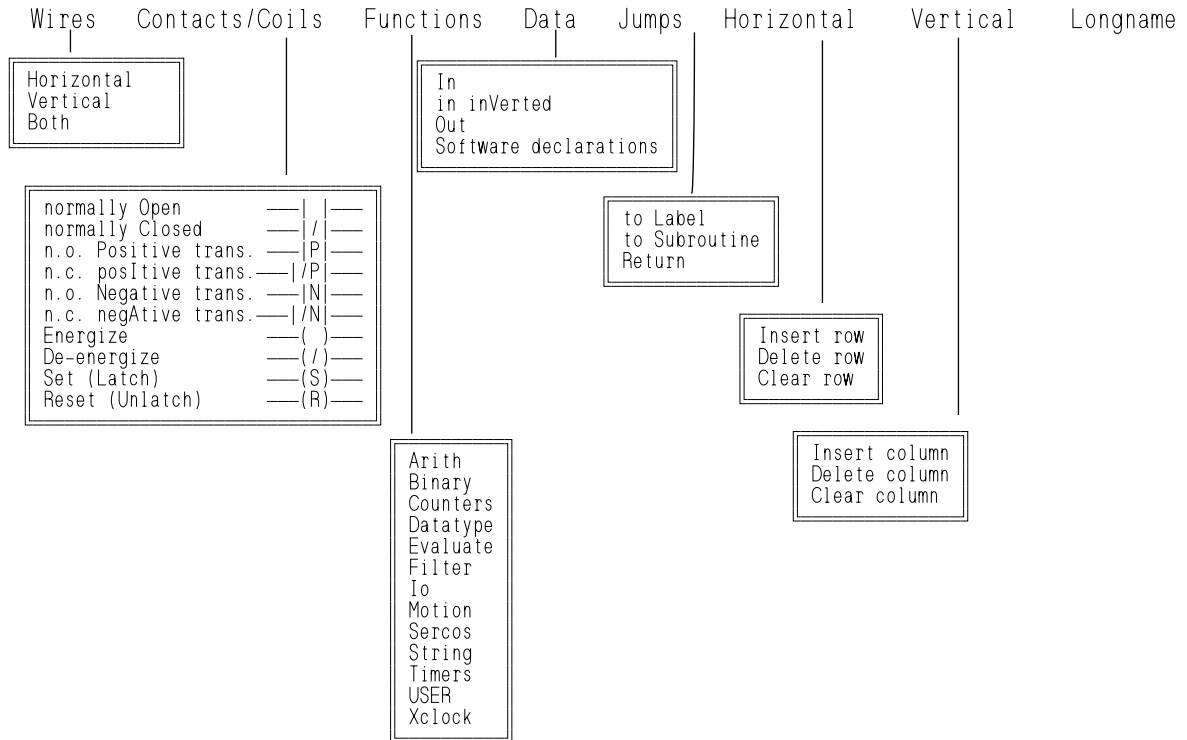
FIGURE 2 - 6. Module Menu



Network Menu

The Network menu is displayed when a network is created (by selecting Network/Insert from the Module menu) or edited (by selecting Network/Edit from the Module menu). From this level elements are added to or deleted from a network. Software declarations can be maintained at this level. Long names can be viewed, edited, or added.

FIGURE 2 - 7. Network Menu



How to Create or Open a Module

A new module is created or an existing one is opened by entering its name. A name is entered by selecting New or Open from under the Module option of the Main menu or Module menu.



A module becomes a DOS file when saved. Its name must conform to MS-DOS standards for filenames. Filenames can have from 1 to 8 characters. Each character must be from this list:

A - Z a - z 0 - 9 \$ % ' _ @ { } ~ ` ! #

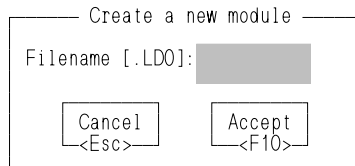
See a DOS manual for additional requirements.

PiCPro automatically adds an *.LDO* (ladder diagram object) extension to the filename. Throughout the programming process several other files, with the same prefix or module name, but different extensions, are created by PiCPro. For a complete list see Appendix A.

1. Access New to create a new module.

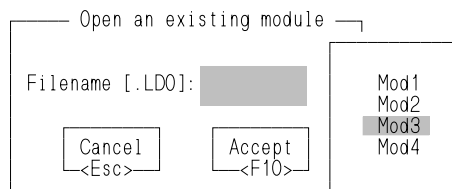


The following screen appears:

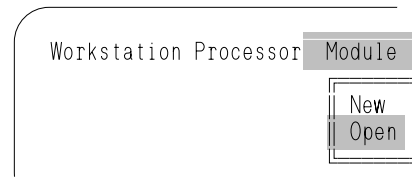


2.Type the name of the new module.

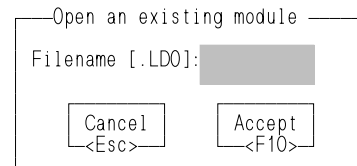
3.Press <F10> to Accept it.



1. Access Open to open an existing module.



The following screen appears:



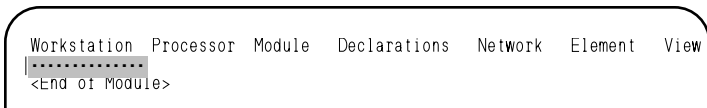
Instead of typing a name, the <F8> key can be pressed to display a list of existing modules. Use arrow keys to move the cursor to a name.

Press <Enter> to select a name.

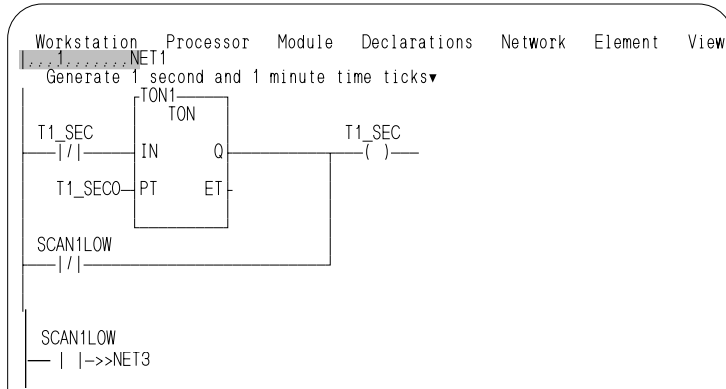
Press <F10> to Accept the name.

NOTE: If New was selected and <F8> is pressed, an existing module will be replaced with a new (empty) module. PiCPro will prompt you before deleting the existing module.

The Module Menu is displayed after a module name is entered. (It can also be returned to when editing a network by pressing <F10> or <ESC>).



If a new module is being created the lower portion of the screen is blank (because there are no networks yet), and the End of Module marker is at the top of the screen. As networks are built (from the Network Menu) this marker moves down, always appearing at the end of the module.



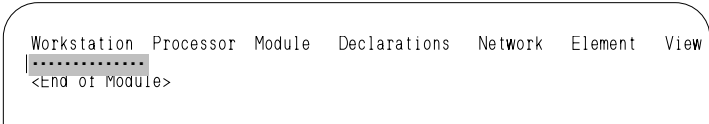
If an existing module is being opened, or if the Module Menu is accessed from the Network Menu, the lower portion of the screen displays the network logic (or a portion of it).

When the Module Menu is displayed you can scroll through the module (the cursor moves from one network number to the next) but you cannot move the cursor inside a network.

When the Network Menu is displayed you can move the cursor throughout the network but you cannot move it to another network. You must exit the network (<F10> or <ESC>) and return to the second Module menu level.

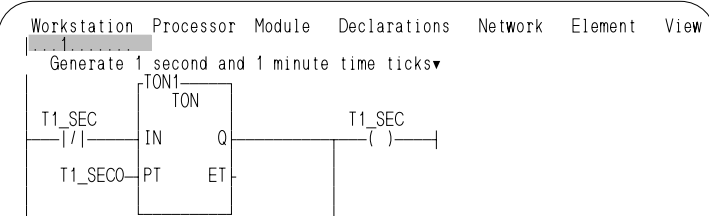
NOTE: All network numbers are controlled (added/changed) by PiCPro. You may assign an eight character label to any network (with the Label command under Network).

The elements of a network are added or changed from the Network Menu. This menu is always accessed from and returned to the Module Menu.



If you are creating a new module, position the cursor as shown in the diagram to the left before accessing the Network Menu.

If you are editing an existing module, position the cursor on the network number you want to edit before accessing the Network Menu.

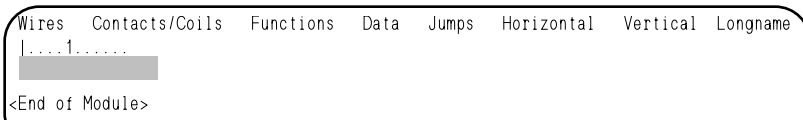


STARTING
PiCPro/PiCServoPro

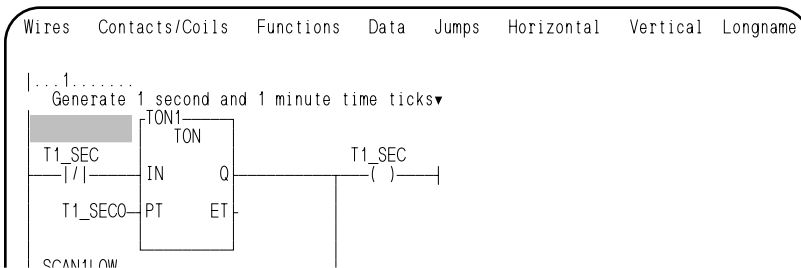
There are two ways to access the Network Menu.

1. Press **<Enter>** when the cursor is positioned as described above.
- or
2. Use the Hot keys **N,E** (Network, Edit)

Workstation Processor Module Declarations Network Element View



When a new module is being created, the Network menu appears, the network is assigned a number, and the cursor locates itself in the upper left-hand cell of the network.



When an existing module is being edited, the Network Menu appears and the cursor moves from the network number to the upper left-hand cell in the network. The cursor can then be moved with the arrow, tab, etc. keys.

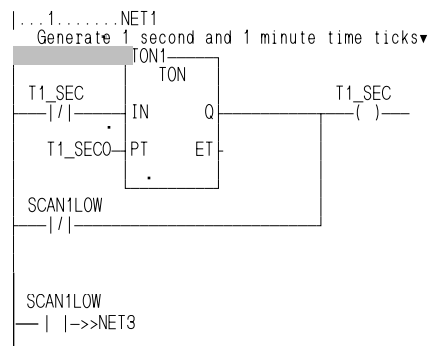
How to add, change, and delete network elements

To add or change an element in a network:

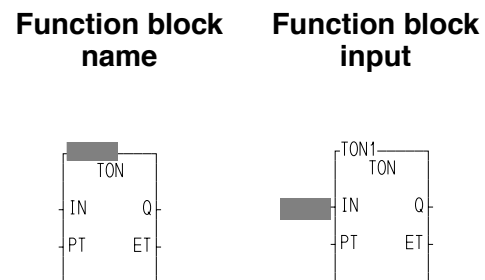
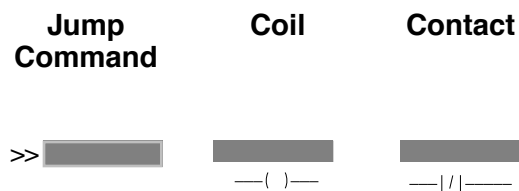
1. Position the cursor at the location where you want the element to be placed.
2. Select the element from its submenu.

PiCPro (re)places the element on the screen at the location of the cursor.

Functions and function blocks should be inserted one or two cells to the right of the left power rail to allow room for entering inputs.



When contacts/coils, functions/blocks, and jumps are selected from the menus, PiCPro positions the cursor to prompt you to enter data for them.



To delete an element, position the cursor on it and press the key.

Cutting and pasting within a network

It is possible to cut and paste network elements while editing a network. Follow the procedure below.

1. Place the cursor on the element you want to cut and paste.
2. Press and hold down the <Shift> key. The selected element will now flash.
NOTE: If the element selected is a function/block, any directly connected inputs and outputs will also be selected.
3. Use the arrow keys to move to the location you want to paste the selected element.
4. Release the shift key. The element will be pasted in the current cursor location.
NOTE: If you want to paste the selected element in more than one location, *do not release* the shift key. Press <Insert> wherever you want the element pasted. If the new location would require that a function/block be deleted, you will be asked to verify that you really want to do this.

How to exit a network

There are two ways to exit from editing a network. Exiting returns you to the Module Menu.

1. Press <F10>. This accepts any changes you made to the network. You must select SAVE or SAVE AS.. under the Module item to save your module to disk.
2. Press <ESC>. This gives you the option to save changes or ignore them. PiCPro displays the message:

Ignore changes and quit editing Network #1
Yes
No

How to create a second network

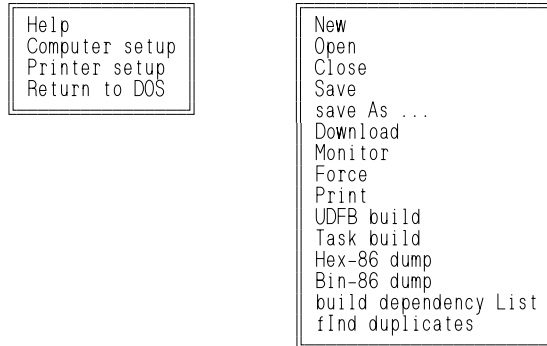
After you have created and saved a network, the cursor positions itself at the end of the network.

Press <Enter> or use the hot keys to reenter the Network (Menu) level. Network number 2 is inserted. The number is assigned automatically by PiCPro.

How to save and close a module

A module is closed and/or saved from under the Module or Workstation menu items.

Workstation Processor Module Declarations Network Element View



There are three ways to save/close a module:

1. Select Close, Save or save As

If Close is selected, PiCPro will ask if you want to save changes.

If Save is selected, PiCPro will save any changes.

If save As is selected, PiCPro will ask you to enter a filename, will save the changes to that file, and make that file the current module.

2. Select New or Open

If New or Open is selected, PiCPro will ask if you want to save changes to the existing module and will close it.

3. Select Return to DOS

PiCPro will ask if you want to save changes to the module.

Designing networks

When you build a module, the capabilities of the PiC900 and the editing features of PiCPro to consider are:

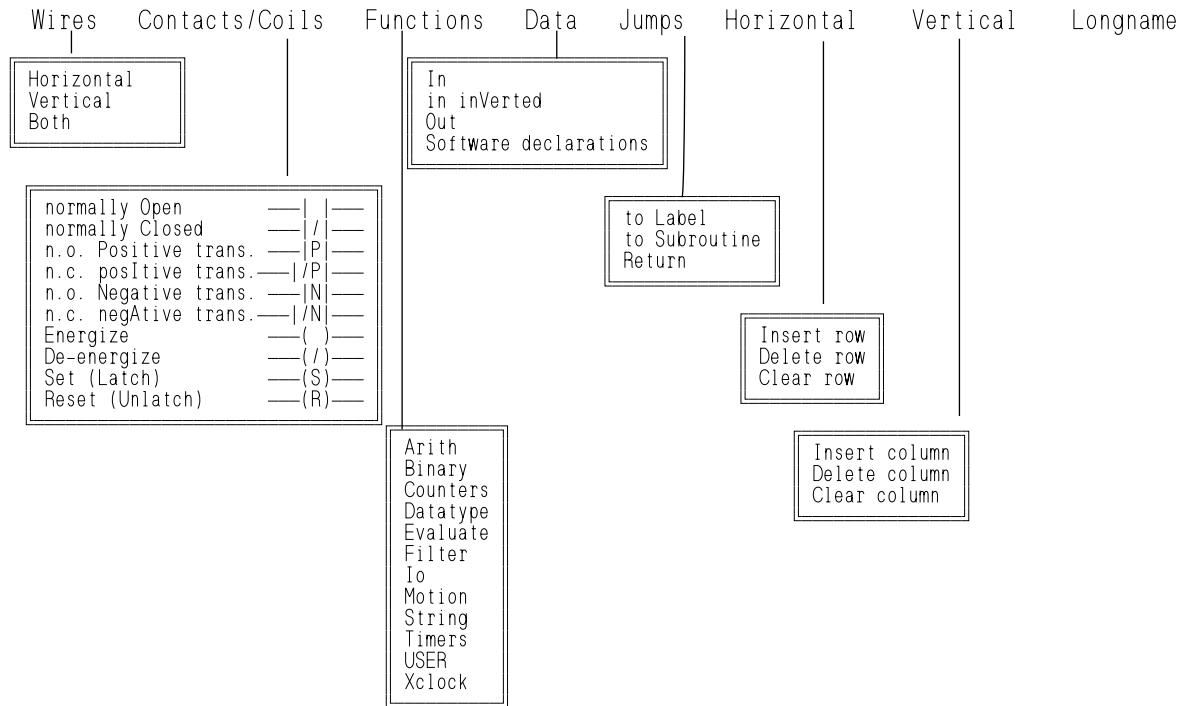
1. Network layout is *free format* -- the number of elements on a rung and the number of branches from a rung are limited only by the 255 cell area size.
2. Commands for editing a module enable you to do the following:
 - search for and/or copy, move, delete entire networks
 - insert networks (to be built) between two networks
 - search for and/or replace elements throughout entire module
 - add up to 100 lines of documentation to each network insert or delete rows and/or columns of network logic.

CHAPTER 3 Programming and Declarations

Chapter 3 provides additional information on programming and declarations. It covers the basic network elements used in creating/editing a network, data types used in software declarations, examples of declaring arrays and structures, and commands for editing and making global changes to a network or module.

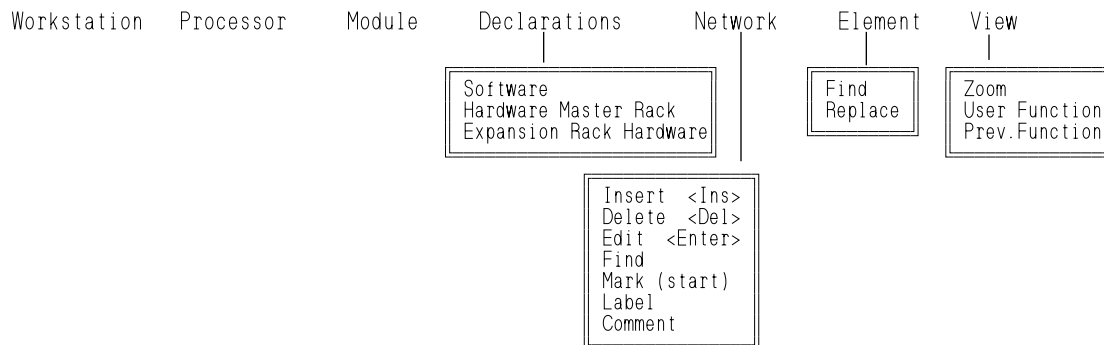
The items on the third level Network menu will be covered first. These are used in creating/editing a network in your module.

FIGURE 3 - 1. Network Menu



In addition, the following items from the second level Module menu will be covered. Chapter 4 also covers some of the items from the Module menu.

FIGURE 3 - 2. Module Menu



Network Menu - Wires

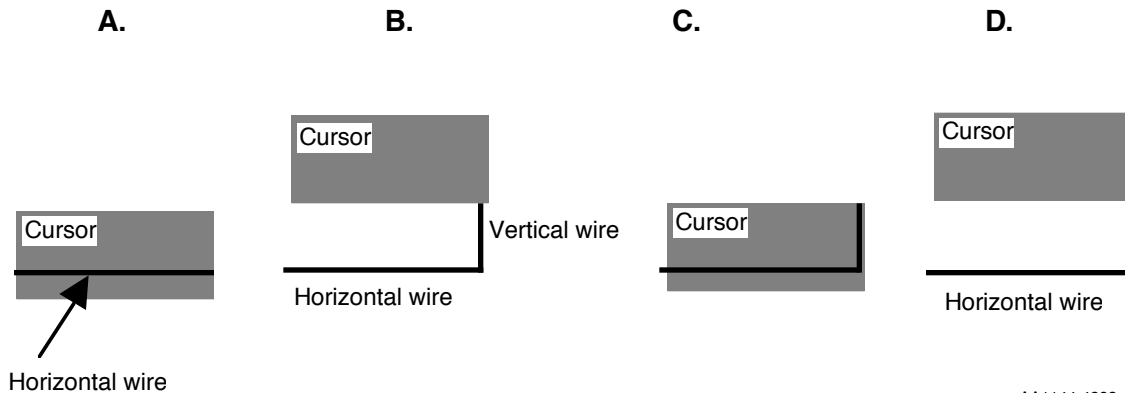
Under Wires you can select three elements - a horizontal wire, a vertical wire, or a combination wire. Use these elements to connect other elements in a network, forming *data paths* along which data is transferred from one element to another. Wires can also be used to broaden or lengthen the spacing of elements on the screen.

Horizontal	–	Provides a horizontal connection between elements in the network
Vertical		Provides a vertical connection between elements in the network
Both	┘	Provides a branch between elements in the network

When making connections within your network, the following keys provide toggling ability for inserting/deleting horizontal and vertical wires.

- <ALT>I** ↑ Inserts/deletes a vertical wire on the right side of the cursor and moves the cursor above the current location.
- <ALT>M** ↓ Inserts/deletes a vertical wire on the right side of the cursor and moves the cursor below the current location.
- <ALT>J** ← Inserts/deletes a horizontal wire on the right side of the cursor and moves the cursor to the left of the current location.
- <ALT>K** → Inserts/deletes a horizontal wire on the left side of the cursor and moves the cursor to the right of the current location.

For example, if the cursor is on a horizontal wire (A) and you press <ALT> I, a vertical wire will be inserted above the horizontal one (B). If you use the down arrow key to move the cursor back down on the vertical wire (C), and press <ALT>I again, the vertical wire will be deleted (D).



AA1144-4892

Network Menu - Contacts/Coils

Under Contacts/Coils you can select the boolean elements listed below. Each boolean element must be assigned a declared boolean variable in the software declarations table. If the element represents a physical input or output, the declaration must specify its location.

In ladder programming these elements symbolize the hardwired circuitry of the system and the variables hold the values that represent their states. The values are always 0 for OFF or deenergized, or 1 for ON or energized. When you insert one of these elements into your network, the cursor locates itself above the item, prompting you to enter the name of the variable assigned to it.

normally Open	— —	If the variable value equals 1, power flow/logic continuity is provided. If the variable value equals 0, power flow/logic continuity is not provided.
normally Closed	— / —	If the variable value equals 1, power flow/logic continuity is not provided. If the variable value equals 0, power flow/logic continuity is provided.
n.o. positive trans*	— P —	If the last write to the variable caused it to go from 0 to 1, power flow/logic continuity is provided; if the last write <u>did not</u> cause the variable value to go from 0 to 1, flow/continuity is not provided or is dropped.
n.c. positive trans*	— /P —	If the last write to the variable caused it to go from 0 to 1, power flow/logic continuity is not provided or is dropped; if the last write <u>did not</u> cause the variable value to go from 0 to 1, flow/continuity is provided.
n.o. negative trans*	— N —	If the last write to the variable caused it to go from 1 to 0, power flow/logic continuity is provided; if the last write <u>did not</u> cause the variable value to go from 1 to 0, flow/continuity is not provided or is dropped.
n.c. negative trans*	— /N —	If the last write to the variable caused it to go from 1 to 0, power flow/logic continuity is not provided or is dropped; if the last write <u>did not</u> cause the variable value to go from 1 to 0, flow/continuity is provided.
Energize	—()—	If power flow/logic continuity to this coil occurs, it is turned on. If power flow/logic continuity to this coil drops, it is turned off.
De-energize	—(/)—	If power flow/logic continuity to this coil occurs, it is turned off. If power flow/logic continuity to this coil drops, it is turned on.
Set (Latch)	—(S)—	If power flow/logic continuity to this coil occurs, it is turned on. If this <u>coil is on</u> and power flow/logic continuity drops, <u>it stays on</u> . If power flow/logic continuity to this coil does not occur, it does not turn on.
Reset (Unlatch)	—(R)—	If power flow/logic continuity to this coil occurs, it is turned off. If this <u>coil is off</u> and power flow/logic continuity drops, <u>it stays off</u> . If power flow/logic continuity to this coil does not occur, it does not turn off.

*Transitional contacts pass power until the next update of the coil. Depending on the logic in your ladder program, this may not be equal to one scan.

Network Menu - Functions

There are several categories of functions/blocks as shown in the table below. Functions and function blocks are the tools which perform operations on data. They perform arithmetic operations, evaluate bits, read and write data, move axes, etc. You specify input values and provide variables into which the PiC900 inserts the output values. The execution of functions and function blocks is always triggered by boolean logic - the wires, contacts/coils described above.

Each function and function block is similar to what is called a subroutine in other programming languages. To use a function/block in your program, simply select one from the menu and connect the inputs and outputs as required. Refer to the Reference Manual for information on functions and function blocks.

Categories of Functions/Blocks

Category	Description
Arith	Perform addition, subtraction, multiplication, division, modulo (remainder), absolute value, square root, transcendental, and negate (opposite) value operations.
Binary	Perform operations on binary data - Logic operations - and, not, inclusive or, and exclusive or; and Bit Shifting/Rotating operations - shift left, shift right, rotate left, rotate right.
Counters	Perform counting operations. They count up to a specified value or count down from a specified value.
Datatype	Convert the representation of data from one type to another.
Evaluate	Compare the values of data. They compare whether values are equal to, not equal to, greater than, less than, greater than or equal to, less than or equal to other values.
Filter	Filter or sort data. They move data, find the maximum of two values, find the minimum of two values, assign a value that is within predefined limits, select a value based on the state of a contact, and select a value based on a numeric value.
Io	Used with the analog in/out, 4-20 mA, thermocouple, RTD, stepper and serial communications modules, the PiC900 communications network, or the data transfer between DOS workstation files, PiC900 RAMDISK files and user port 2 on the PiC900.
Motion	Control axial movement.
Sercos	Communicate serially in real-time between PiC and drives.
String	Operate on STRINGs of data. They find the length of a string, or find, delete, insert, extract, concatenate, or replace characters in a string.
Timers	Energize or deenergize outputs after a duration of time.
USER	Appears only after the user defined function block or task features have been used. Any UDFB or TASK you create will be accessed through USER.
Xclock	Perform clock and calendar functions. They extract the time or day of the week from the PiC900, or assign a time to a variable. Can set up a servo clock to use with TASKs when no servos are programmed.

The PiC900 processes a function or function block left to right - first the inputs are gathered, then the operation is performed, and then the outputs are written. The only time a function/block does not execute is when the logic at EN does not trigger it.

NOTE

Output variables will have unpredictable values and the output at OK will not be energized when:

1. An output variable does not have enough bits to hold the result.
2. An output variable is an unsigned integer (recognizes only non-negative numbers) and the result is negative.
3. The operation attempts to divide a number by zero.
4. The input data is invalid.
5. Floating point number results are infinite, indefinite or not-a-number.

Functions and function blocks are very similar to each other and they look similar on the screen. The one important difference between them is that function blocks may take more than one scan to complete an operation, whereas functions must complete an operation synchronously with the scan.

This requires that function blocks have internal storage for their variables from scan to scan until they complete the operation. You must declare function blocks and assign each one a unique name so PiCPro can reserve memory for them. Functions do not require this because they finish executing with the scan.

Function/Block Format

Functions and function blocks appear as rectangles on the screen. The labels inside the boxes are not changeable. They tell you the inputs and outputs to enter, and show you where to connect them.

Diagram of a Function

The inputs to the function are on the left side.

EN is where you connect the boolean logic that will "enable" or trigger execution of the function

IN is where you connect the input variable that is to be operated on

A title, the name of the function, appears at the top.

The outputs of the function are on the right side.

OK is where you connect logic that will have power flow to it as soon as execution has finished (successfully)

OUT is where you connect the variable that will hold the result of the operation.

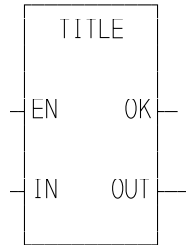
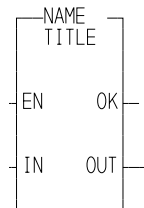


Diagram of a Function Block

A function block has the same general appearance as a function, but it also shows the unique name you assign to it in software declarations.

NOTE: Function blocks maintain internal storage. Therefore, when the enable is removed, the outputs typically will not change.



NOTE: Some functions/blocks can have several inputs/outputs.

Function/Block Inputs/Outputs

Each function/block has only certain types of data it accepts as inputs or provides as outputs. PiCPro will flag you during the programming process if you attempt to assign input/output data that is not acceptable for a function.

If a required input is missing, PiCPro will flag it when you attempt to exit (<F10>) a network.

Connections to outputs are optional. Output variables do not need to be connected to functions/blocks if you do not want to receive that output data.

Boolean Inputs/Outputs

Boolean inputs and outputs are wires, contacts, control relays, or coils. Every function/block has a boolean input on the upper left side and a boolean output on the upper right side. Although these inputs (or outputs) may have different labels, they function the same for every function/block:

	Boolean Inputs		Boolean Outputs
EN (enable) CD (count down) CU (count up) REQ (request)	These inputs (on the upper left side) are always used to start execution of the function/block. The logic to this point must be connected and must be satisfied or the function/block is not executed.	OK DONE Q	These outputs (on the upper right side) serve as a signal that the function/block has finished executing and that the execution was successful. The ladder logic connected to this point will be satisfied immediately if and when the task has been completed and no errors have occurred. The use of OK is optional but it can serve as a troubleshooting tool. It does not energize if an error occurs, such as an overflow (output variable does not have enough bits to hold the result).

Some functions/blocks have more than one boolean input/output. Their usage depends on the function/block. They might initialize a counter, trigger a timer, tell that an axis is in position, etc.

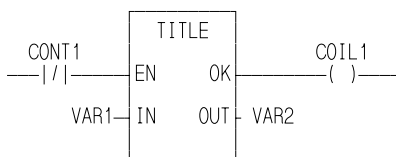
Non-boolean inputs/outputs

The non-boolean inputs are constants or variables. They are either the data that is to be operated on or they provide data required to perform the operation, such as a number of counts.

The non-boolean outputs are variables. They either hold the result of the operation, or hold other pertinent information about the operation.

Connection requirements

Inputs to and outputs from functions/blocks must be connected to the functions/blocks.



To connect non-boolean elements use the Data menu. Select In or inVerted to enter an input to a function/block. Select Out to enter an output for a function/block.

To connect a boolean element position the cursor at the input/output point, then select the element from the Wires or Contacts/Coils menu.

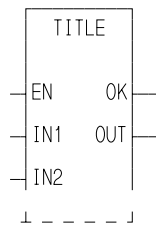
Extensible functions

Some of the functions can be "extended" to accommodate more inputs. For example the addition function lists two inputs on the left side of the rectangle, but the function can accept and add up to 17 numbers.

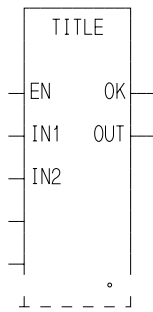
You can lengthen the box to enter more inputs (by positioning the cursor on the lower left side of the rectangle, and selecting Data In). All extensible functions can have up to seventeen inputs. Once a function has been extended, it cannot be shortened.

Functions that are extensible are displayed on the screen with dashed lines outlining the lower border (instead of solid lines).

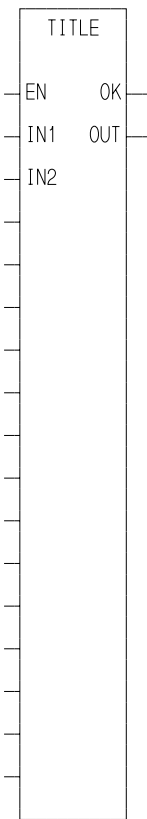
Extensible Function



Extended Function



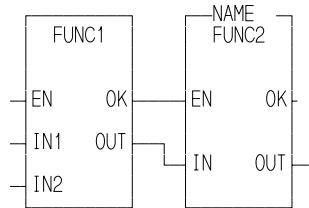
Fully Extended Function



Chaining Functions/Blocks

Functions/blocks can be linked or chained by data paths. In the following diagram two links are shown.

1. OK is linked to EN. The successful execution of FUNC1 triggers the execution of FUNC2.
2. OUT is linked to IN. The output value of FUNC1 is passed directly to FUNC2, with no intermediate variable holding the value.



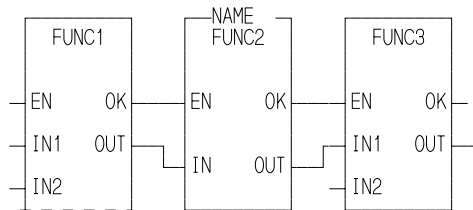
Any number of functions/blocks can be chained and a variety of connections can be made. The requirements for chaining functions/blocks are:

1. If the output of a function block serves as an input to another function/block, it must be the same type.
2. Outputs to inputs must flow left to right.
3. A wire must be used to connect the outputs of one function to the inputs of another. They cannot be simply positioned side-by-side.

The diagram on the left illustrates correct chaining. The chaining on the right will not work.

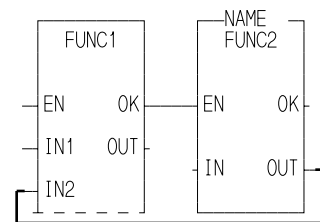
Acceptable chaining

Left to right data flow and
 inputs/outputs are the same types



NOT acceptable chaining

No right to left data flow



Network Menu - Data

In the data category, there are four choices as shown below.

In	Selected to enable the connection and entering of inputs to functions/blocks.
in inVerted	Selected to enable the connection and entering of inverted inputs to functions/blocks.
Out	Selected to enable the connection and entering of outputs to functions/blocks.
Software declarations	Calls up the declarations table. Allows access to the software declarations while creating or editing networks.

All data elements that can be read from are in the form of variables or constants. All data elements that can be written to are in the form of variables.

Variables serve three purposes:

1. They provide input values to functions/blocks.
2. They receive output values from functions/blocks.
3. They hold the values which represent the states of contacts, control relays, and coils.

Variables must be created in the software declarations table. In that table you can do the following:

1. Assign each variable a name.
2. Tell what kind of data it represents, called its *data type*.
3. Change its initial value - optional, PiCPro initializes all variables to zero.
4. Tell its hardware module location, if it represents a physical input or output.
5. Make it retentive, global, or external - optional, retentive makes it retain its value upon a warm restart or power-on. (Global and external apply to attributes shared with other LDOs that can be merged or with TASKs.)
6. Modify an attribute as an input or output to a user defined function block if the module is to be converted to a UDFB.

Constants provide input values to functions/blocks. They cannot be entered anywhere else in the network.

Constant values must be entered in the same form and within the same range that a variable value would be entered in the software declarations table.

There are several kinds of variables that can be created. Also, variables can be manipulated in special groups called structures and arrays. All variables are in one of the following categories.

Bitwise

Bitwise variables/constants are used to represent binary values. When manipulated, these types of data are treated as series of binary digits.

Data type	# of bits	Range	Description
BOOL boolean	1	0 or 1	A boolean value is a bit. Contacts, coils, and control relays are always represented by boolean <u>variables</u> , where: 0 = off/no, 1 = on/yes. Byte, word, double word, and long word values are groups of bits that are used when logical and bit manipulation operations are performed.
BYTE	8	eight 0s-1s	
WORD	16	sixteen 0s-1s	
DWORD double word	32	thirty-two 0s-1s	
LWORD long word	64	sixty-four 0s-1s	

Formatting values

Bitwise values can be entered in binary, octal, decimal, or hexadecimal. The format for entering these values is shown below for the number 23. (2#, 8#, and 16# are required.)

binary - 2#10111 octal - 8#27 decimal - 23 hexadecimal - 16#17

NOTE: Bitwise constants are limited to 32 bit values. A constant can be entered for any bitwise data type. For data types 32 bits or less, the constant value must be within the range of the data type. For data types greater than 32 bits, the 32 bit constant value will be extended. Zeros will be placed in the 32 most significant bits. The constant value will be held in the least significant 32 bits.

When booleans are entered as constants, enter the following.

For	Enter
On	(hex) 16#80 or (decimal) 128
Off	0

For boolean initial values, enter a 0 or 1.

Numeric

Numeric variables/constants are used to represent integers and real or floating point numbers. When manipulated, these types of data are treated as numbers.

Data type	# of bits	Range	Description
SINT short integer	8	-128 to +127	Integers are whole numbers or zero.
INT integer	16	-32,768 to +32,767	
DINT double integer	32	-2,147,483,648 to +2,147,483,647	
LINT long integer	64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
USINT unsigned short integer	8	0 to 255	Unsigned integers are greater than or equal to zero. PiCPro will prevent you from entering a negative number for an unsigned integer.
UINT unsigned integer	16	0 to 65,535	
UDINT unsigned double integer	32	0 to 4,294,967,295	
ULINT unsigned long integer	64	0 to 9,223,372,036,854,775,807	
REAL	32	significant digits 6-7	Real numbers are floating point numbers. They contain a decimal point and an exponent indicating the power of ten the number is to be multiplied by to obtain the value represented.
LREAL long real	64	significant digits 15-16	

NOTE: You must have a math coprocessor installed on your PiC900/90 CPU module to perform any functions involving any 64 bit registers, logarithmic, exponential, trigonometric, and floating point mathematical operations.

Formatting values

Numeric values can be entered in binary, octal, decimal, or hexadecimal. The format for entering these values (same as bitwise) is shown below for the number 35. (*2#*, *8#*, and *16#* are required.)

binary - *2#*00100011 octal - *8#*43 decimal - 35 hexadecimal - *16#*23

NOTE: Numeric constants are limited to 32 bit values. A constant can be entered for any numeric data type (SINT, INT, LINT, etc.). For data types 32 bits or less, the constant value must be within the range of the data type. For data types greater than 32 bits, the 32 bit constant value will be extended appropriately.

String

String variables are used to represent a sequence of zero or more characters. The characters are ASCII and/or extended ASCII.

Data type	Length in bytes (Internal)	Range	Description
STRING	Two plus number of declared characters	0 - 255 ASCII characters	String type variables are used to read or write messages. Every string has 2 extra bytes that hold information about the string. The 1st byte tells the maximum or declared length of the string. The 2nd byte tells the actual length of the string. NOTE: STRINGs cannot be entered as constants.

Formatting values

String values are keyed in exactly as they are to be interpreted. A 3 character combination of the dollar sign (\$) followed by 2 hexadecimal digits is interpreted as the hex representation of an 8-bit ASCII character code. A 2 character combination of the dollar sign followed by a character is the ASCII interpretation given below:

<u>Character string</u>	<u>Interpretation</u>
\$\$	dollar sign
'	single quote
\$L	line feed
\$N	new line
\$P	form feed (page)
\$R	carriage return

Time:time of day

Time of day values represent the time of day, the date, or both.

Data type	# of bits	Range	Examples	Description
DATE	16	Jan. 1, 1988 - Dec. 31, 2051	Year/month/day D#1992-12-31	Date and time values are used when the date or time is to be printed, or when an event is to be triggered at a given moment in time. Values can be assigned manually or extracted from the PiC900 clock (with functions).
TIME_OF_DAY	32	00:00:00 - 23:59:59	Hours/minutes/seconds TOD#23:59:59	
DATE_AND_TIME	32	same ranges as above, combined	Year/month/day/hours/ minutes/seconds DT#1992-12-31-23:59:59	

Months are numbered 1 - 12. Days of the month are numbered 1 - 31.

Hours are numbered 0 - 23. Minutes are numbered 0 - 59. Seconds are numbered 0 - 59.

Formatting values

Values are entered in the following formats. (*D#*, *TOD#*, *DT#* , colons (:)) and dashes (-) are required)

DATE - D#1990-09-26

TIME_OF_DAY - TOD#16:26:23

DATE_AND_TIME - DT#2009-08-05-02:01:31

(Sept. 26, 1990)

(4:26:23 p.m.)

(Aug. 5, 2009, 2:01:31 a.m.)

Time:duration

A time (duration) value represents an amount of time.

Data type	# of bits	Range	Example	Description
TIME	32	0 - 49d17h2m47s295ms 0 - 1193h2m47s295ms 0 - 71582m47s295ms 0 - 4294967s295ms 0 - 4294967295ms	Days/hours/ minutes/seconds/milliseconds T#1d2h3m4s5ms	Time duration values serve as inputs to timer functions for counting up/down for a specified amount of time; and they serve as outputs into which elapsed time values are entered.

d - day h - hour m - minute s - second ms - millisecond

Formatting values

Values are entered in the following formats. (*T#* is required, and *d*, *h*, *m*, *s*, and *ms* are required if the value for that increment is not zero)

T#1d3h7m16s45ms T#14d T#459871ms

Function/Block Data

Constants can be inputs to any functions/blocks except functions that operate on STRINGS (string values must be put into variables). A constant value must be in the range and format it would be in if it were a variable value.

Generally, all input variables and the output variable must have the same data type. However, this is not true for a function whose purpose is to change a data type i.e., converting numeric type data into bitwise type data. It is also not true for inputs that are providing extraneous data for the operation, and outputs that are providing information about the operation. In all other cases though, the output data type must match the input data types.

Although input and output data must be of the same type, input and output variables usually do not have to be unique variables. For instance, you could add the constant 1 to a variable called VAR1, and place the result in VAR1.

IMPORTANT

If an output variable is unique from an input variable, the input variable is unaltered by execution of the function/block. Only the output variable is changed.

Functions which return a string type variable as an output have a unique characteristic. The output variable must be assigned on the input (left) side of the function. This is necessary because STRINGS require memory allocations before the operation is performed.

Data Groups of Variables - Arrays and Structures

Variables can be grouped to create entities called arrays and structures. This ability to group data enables you to keep various types of data together that have a common link. Values can be read into or written from these groups with I/O functions. Also, individual variables in structures and arrays can serve as inputs to and outputs from functions/blocks.

More information on arrays and structures can be found in the section on software declarations later in this chapter.

A *jump* command causes execution of the ladder to go to or "jump" to a designated network in the program. The network being jumped to must have a label assigned to it by you using the Label command under Network. Jumps are always made to the beginning of a network.

When the jump "to Label" or "to Subroutine" command is selected, the Jump command is inserted into the network. You specify which network by supplying the label when you enter the command.

When you jump to a label, you cannot return. When you jump to a subroutine, you can use the Return command to return to a specific network in your ladder.

to Label	Places the symbol for the Jump to label command (>>) into the network and prompts you to enter the label of the network to which execution should jump.
to Subroutine	Places the symbol for the Jump to subroutine command (<) into the network and prompts you to enter the label of the network to which execution should jump.
Return	Places the symbol for the Return command (<return>) into the network. Can be placed in a subroutine where it returns you to the network from which the jump originated or in the main module (typically at the end of the program before any subroutines) where it takes you to the <End Of Module> or in a UDFB source ladder (typically at the beginning) where it takes you to the <End of Module> if the enable (EN) is not set.

The diagram that follows illustrates how your program executes with *jump to subroutine* and *return* commands.

Notice that a return command is placed not only at the end of the subroutine, but also at the end of the main module directly before the subroutine network. The return at the end of the subroutine takes you back to the original jump to subroutine command network. The return at the end of the main module program takes you to the end of the module. If it were not included, the program would execute the subroutine again.

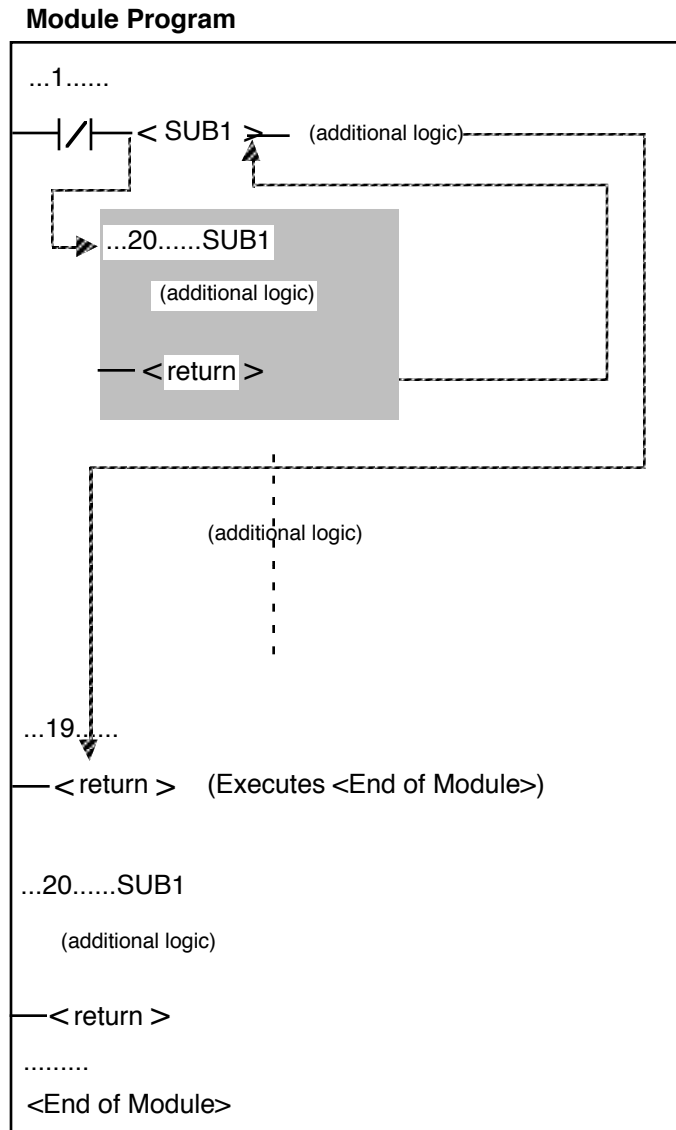
It is recommended that subroutines be placed at the end of your main module program.

Execution Path for Subroutine/Return Commands

If the jump to SUB1 is activated (network 1), the program jumps to the network labeled SUB1 (network 20). It will execute the logic in subroutine 1.

When it reaches the return command in SUB1 (network 20), it returns to network 1. It executes any additional logic to the right and/or below the subroutine command.

When it reaches the return command at the end of the main program (network 19), it executes the end of the module function at that point.



Network Menu - Horizontal

- Insert row** This option inserts an empty row into the network, lengthening any vertical wires in the row. The row gets inserted above the line that the cursor is on.
- Delete row** This option deletes the row the cursor is on and moves every row below it up one line.
- Clear row** This option clears (deletes) the row the cursor is on, leaving an empty row. The rows below the cleared row do not move up.

NOTE: You cannot insert/delete/clear a row if it contains a function/block cell.

Network Menu - Vertical

- Insert column** This option inserts a column into the diagram to the left of the cursor, widening the diagram with horizontal wires. All columns to the right of the cursor shift right.
- Delete column** This option deletes the column the cursor is located on and shifts all columns (to the right of the cursor) left.
- Clear column** This option clears (deletes) the column the cursor is on. All columns to the right of the cursor do not shift, leaving a "cleared" column.

Network Menu - Longname

The Longname command allows you to view/edit longnames. Longnames can have up to 40 characters on four lines. You can view, edit, or add a longname to a variable in a network. Place the cursor on the variable and press **L** (Longname). When you add or change a longname for a variable in the network, PiCPro updates the software declarations longname field automatically.

Longnames will appear on your screen when you use the **View Zoom** command.

Module Menu - Declarations

When in the Module menu level, the Declarations menu allows you to declare software, hardware master rack, and expansion rack declarations. Software declarations are also available when in the Network menu level under the Data menu.

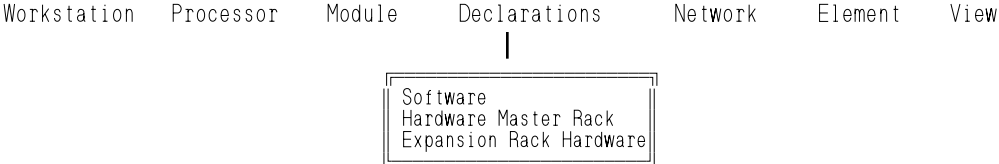
Hardware Declarations - Master and Expansion racks

Hardware declarations define what type of hardware module occupies each *slot* in all PiC900 racks.

The hardware configuration defined by the declarations tables is compared to the software configuration when the module is downloaded. PiCPro can detect an error such as an output that is defined at an input location. A message will be displayed that describes the error when you issue the download command.

```
OUT3 calls for output 3 in slot 6
which is not an output!
Press any key to continue
```

There is only one way to call up a hardware declarations table. Select **Hardware Master Rack** from under the **Declarations** menu.



The Master Rack Hardware Declarations box lists the PiC900 *slots*, corresponding to the position in the rack as shown below.

```

Workstation Processor Module Declarations Network Element View
Master Rack Hardware Declarations

Press <F8> for hardware list, then
select desired type and press <Enter>.
Slot 1: CSM
Slot 2: PiC900, 64K
Slot 3: Empty
Slot 4: Empty
Slot 5: Empty
Slot 6: Empty
Slot 7: Empty
Slot 8: Empty
Slot 9: Empty
Slot 10: Empty
Slot 11: Empty
Slot 12: Empty
Slot 13: Empty
Expansion I/O Type:None

Cancel Accept
<Esc> <F10>

```

IMPORTANT

Slots 1 and 2 are always dedicated to the CSM and the CPU modules.

To declare your hardware modules in the Master Rack Hardware Declarations table:

1. Position the cursor on the number.
2. Press <F8> to display the list of modules available.
3. Cursor to the appropriate group and then module.
4. Press <Enter> to make a selection.
5. Repeat steps 1 - 4 to enter each module.
6. If you are doing I/O expansion with either rack I/O modules or block I/O modules, you must indicate this in the last line of the Master Rack Hardware Declaration table before entering information in the Expansion Hardware table.
7. Press <F10> to Accept the configuration.

To declare Rack I/O Expansion racks:

After selecting *Rack I/O* in the Hardware Master Rack table, select Expansion Hardware from the Declarations menu.

In the box that appears, enter the rack number. Expansion racks are numbered 1 to 7 ; #1 is the rack connected to the master or CPU rack, #2 is connected to #1, etc. Press <F10> to Accept

Follow the steps (excluding step 6) above for declaring the modules in the Master Rack.

To declare Block I/O Expansion racks:

After selecting *Block I/O* in the Hardware Master Rack table, select Expansion Hardware from the Declarations menu.

The Define Block I/O Hardware Configuration table appears. Block I/O expansion modules are numbered 1 to 77; #1 is the module connected to the master or CPU rack, #2 is connected to #1, etc.

Follow the steps (excluding step 6) above for declaring the modules in the Master Rack.

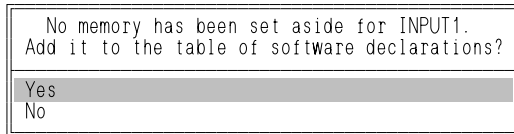
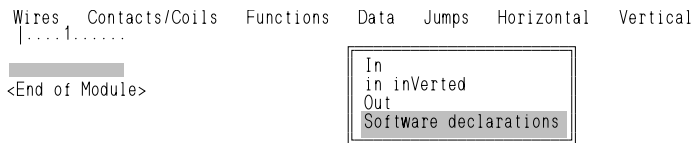
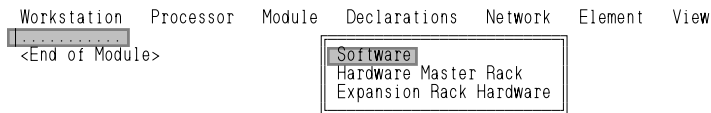
IMPORTANT

Slots 1 and 2 are always dedicated to the CSM and the I/O driver modules.

Software Declarations

A complete software declarations table must have the names and data types of every variable, contact, coil, function block, structure and array that is in the module. It also must have the I/O locations for all physical inputs/outputs. Also, variables are given (different) default values, made retentive, and given long names through software declarations.

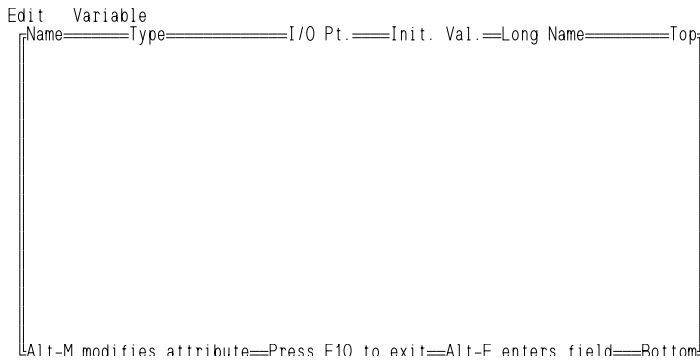
There are three ways to call up the software declarations table.



1.The table can be called up from the Module Menu. Usually this menu is used if you are making declarations before entering the ladder logic. But it can be used at any time when this menu is displayed.

2.The table can be called up from the Network Menu. Usually this menu is used if you want to edit the table while entering the ladder logic.

3.The table can be called up by responding "yes" to this prompt. This is the message that is displayed when you enter an element into the network that has not yet been declared (and must be declared).



The software declarations table is shown on the left. There are two menus that are explained in the sections that follow.

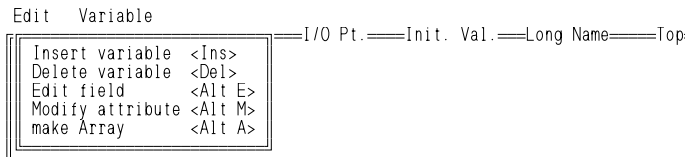
The following will help you move around within the declarations table.

<F10> or <ESC> Saves the table and exits the screen. The only time an entry (in any field) is not saved is when <ESC> is pressed immediately after an entry has been typed.

To move the cursor from:

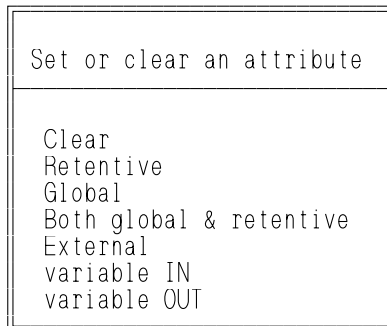
- Top -displays on the right at the top or beginning of the table
- Field to field -with the arrow, tab, or enter keys.
- Line to line -with the arrow, tab, or enter keys.
- Page to page -with the page up and page down keys.
- Top to bottom -with the end key, pressed 3 times.
- Bottom -displays on the right at the bottom or end of the table.
- Bottom to top -with the home key, pressed 3 times.

Software Declarations - Edit Menu



The Edit Menu is shown on the left.

- EI or <Ins>** The Insert variable command allows you to insert a new line before the line the cursor is on.
- ED or ** The Delete variable command allows you to delete the current line when the cursor is on the Name field. If the cursor is anywhere else in the line, the field contents will be deleted.
- EE or <Alt-E>** The Edit field command allows you to edit the selected field. The cursor can be moved from character to character within the field.
- <ALT-D>** Deletes an entire field when editing.
- <ALT-U>** Undoes editing changes.
- EM or <Alt-M>** The Modify attribute command allows you to modify the attribute of the selected variable. To assign an attribute, position the cursor on the same line as the variable (in any field), call up the list, then select an item (with hot keys or with arrow keys and <ENTER>).



Clear - Clears or removes a variable's attribute.

Retentive - Makes a variable retentive (retains its value upon a warm restart or power cycle, but not on a cold restart). Physical I/O points and individual structure members cannot be made retentive. All other elements, including entire structures, can be made retentive.

Global - (Used only with optional LDOMerge software package.)

Applied when the LDO is to be merged with one or more other LDOs. It identifies the variable as the master or global variable. Any variable with the same name in other LDOs will inherit this variable's definition (data type, I/O point, initial value) when the LDOs are merged.

Both global & retentive - Defines the variable as the global variable and makes the variable retentive. See descriptions above for retentive and global. It should be applied only when the LDO is to be merged with one or more LDOs.

External - (Used with the optional LDOMerge software package or with TASKs.)

Applied to variables when an LDO is to be merged with one or more other LDOs. It signals PiCPro that this variable is used in more than one LDO, and that the variable's global definition is in another LDO. If the properties of this variable (data type, I/O point, initial value) differ from the properties of its global counterpart, the variable will acquire the properties of the global variable when the LDOs are merged.

Applied to a TASK variable that is to be shared with other tasks.

NOTE: Only mark the variable as External in the TASK LDO, never in the main LDO in which the TASK is called.

variable IN - Designates the variable as an input to a user defined function block. When the LDO module you are creating will be converted to a UDFB, every variable you want to use as an input to the UDFB must have this attribute.

NOTE: The first variable with the "I" attribute in the declarations table will become the EN input of the UDFB. Its data type must be BOOL. Additional UDFB inputs should be entered in the order you want them to appear on the left side of the function block.

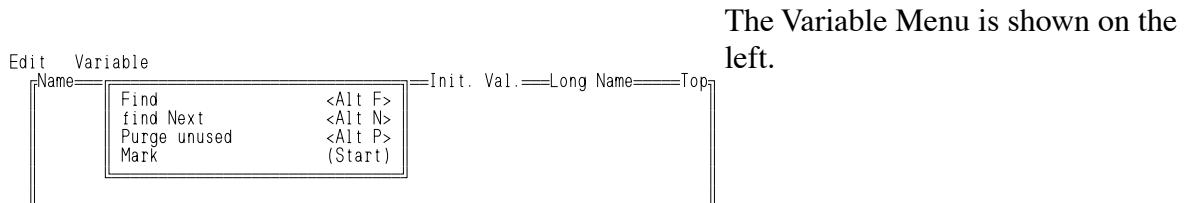
variable OUT - Designates the variable as an output to a user defined function block. When the LDO module you are creating will be converted to a UDFB, every variable you want to use as an output from the UDFB must have this attribute.

NOTE: The first variable with the "O" attribute in the declarations table will become the OK output of the UDFB. Its data type must be BOOL. Additional UDFB outputs should be entered in the order you want them to appear on the right side of the function block.

EA or **<Alt-A>** The make Array command allows you to convert a variable to a multi-element array or change the array size. To use the Array command, highlight the Type field and press **EA** or **<Alt A>**.

Array size can range from 1 to 999 elements.

Software Declarations - Variable Menu



VF or <ALT-F> The Find command finds an entry by the whole name or a portion of the name, by the variable type, or by an I/O point. Enter the appropriate data in the Find name, Find type, or Find I/O point box that appears when you select one of the following

Find
Name or portion of name variable Type I/O point

VN or <ALT-N> The find Next command finds the next occurrence of the name, type, or I/O point from the last Find command. The search starts with the line after the selected line.

VP or <ALT-P> The Purge unused command allows you to purge unused variables from the software declarations table.

Delete unused structure elements?
Yes No

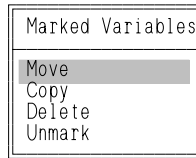
VM The Mark command allows you to mark, copy, delete, and move variables.

1. Locate the cursor on the variable in the declarations table you want to mark.
2. Select the Mark (start) command.

The variable gets marked (highlighted) and the menu changes to Mark (end). To mark more than one variable scroll the screen. Do not move the cursor if you want only one variable marked.

3. Select Mark (end).

The screen displays the following prompt:



4. Select Move, Copy, Delete or Unmark.

Move The screen displays the following prompt at the end of the marked selection
Move here <Enter>, or quit <Esc>

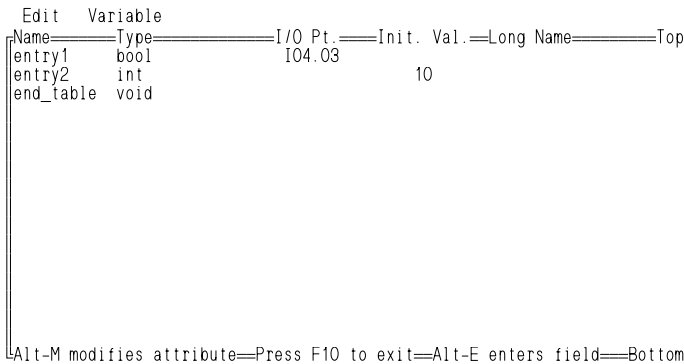
Scroll the screen to the location where you want the marked variable(s) to be moved. The prompt moves too. Press <ESC> to exit - unmark the variable and not move it. Press <Enter> to move the variable. It gets deleted from its previous location and inserted in front of the variable that is highlighted by the prompt.

Copy Works identically to the Move command except that the variable is copied. The original variable remains in place and a copy of it is placed at the location of the prompt.

NOTE: You must change the name of each copied variable to prevent a conflict with existing variables.

Delete Asks if you want to delete marked variables. Answer Yes or No.

Unmark Unmarks a variable, and exits out of the marking procedure.



The five types of data that are entered are listed at the top of the table. All data for an element is entered in one row, spread into columns.

NOTE: The first time the table is called up, the cursor is on the end_table entry. Simply start typing and that entry will jump to the next line.

Name — This must be the first entry made for an item.

entry1	
entry2	
end_table	

Enter from 1 to 8 characters:

- 1st character - must be alpha (A - Z)
- 2nd - 8th characters - can be alpha (A - Z), numeric (0-9), or underscore (_)

If a name is changed in this field, every occurrence of the variable in the module is changed.

Type

```

=====Type=====
      BOOL
      INT
  
```

This is the *type* of function block or the *data type* of a variable. The Type is selected from a list.

PiCPro enters a type for every element after a name has been entered. The type is either boolean (default) or the type of the previous element. This is convenient if you are entering several like items. For example, if you are entering several physical outputs, PiCPro will enter boolean as the type for each one, and number the output locations consecutively.

To change the type that PiCPro enters:

1. Position the cursor in the Type field
2. Press any key to display the list*
3. Scroll to a type

- * The list can be called up by keying the first letter of the type. The list will display starting with that element. If two or more types start with the same letter, call up the list by typing the 1st, 2nd (and 3rd, etc.) letters.

Function BlockTypes

Call up a list of function blocks by pressing F when the cursor is in the type field (or select function block from the list of types). Select a function block by moving the cursor to one and pressing <ENTER> or by using hot keys.

The selected function block displays behind the characters <fb>.

```

Name=====Type=====
|| TON1      <fb>TON
  
```

String Types

If STRING is selected as the data type, PiCPro asks how many characters are to be in the string.

1. Type in a number.
2. Press <F10>.

The number appears in brackets [] behind the word STRING.

```

Name=====Type=====
|| TON1      STRING[9]
  
```

I/O Pt.

==I/O Pt.==
I04.03
INT

The I/O point is for physical inputs and outputs only. It defines the rack and slot location of the hardware module, and the channel location of the input or output. An input/output module must be declared for the slot in the hardware declarations table. The data type of the element must be boolean (or the cursor will not move to this field).

When using Block I/O modules, the I/O point entered must begin with BI for block input or BO for block output. Include the block number and the point number separated by a decimal point.

The master or CPU rack is #0. Expansion racks are numbered 1 - 7, where #1 is the rack connected to the master, #2 is the rack connected to #1, etc. Slots are numbered left to right when facing the PiC900. Slot 1 and slot 2 are reserved for the CSM/CPU module. On an expansion rack, slot 2 is reserved for the I/O Driver module.

Master Rack

Enter 4 to 6 characters

- 1st - I or O Input or Output
- 2nd - 0 - 1 First digit of module slot number (omit if 0*)
- 3rd - 0 - 9 Second digit of module slot number
- 4th - . (point) Used as a separator
- 5th - 0 - 3 First digit of channel number (omit if 0*)
- 6th - 0 - 9 Second digit of channel number

Expansion Rack I/O

Enter 5 to 8 characters

- 1st - I or O Input or Output
- 2nd - 1 - 7 Expansion rack number (1 - 7)
- 3rd - . (point) Used as a separator
- 4th - 0 - 1 First digit of slot # (omit if 0*)
- 5th - 0 - 9 Second digit of slot #
- 6th - . (point) Used as a separator
- 7th - 0 - 3 First digit of channel number (omit if 0*)
- 8th - 0 - 9 Second digit of channel number

Expansion Block I/O

Enter 5 to 7 characters

- 1st/2nd - BI or BO Block Input or Block Output
- 3rd - 0 - 1 First digit of module number (omit if 0*)
- 4rd - 0 - 9 Second digit of module number
- 5th - . (point) Used as a separator
- 6th - 0 - 3 First digit of point number (omit if 0*)
- 7th - 0 - 9 Second digit of point number

*Zeros are entered automatically by the software in these cases.

Examples

If the input is in master rack at slot 4, channel 3, you enter:

I4.3

If the output is from expansion rack # 7 at slot 12, channel 10, you enter:

O7.12.10

If the input is in block I/O module 33, point 5, you enter:

BI33.5

BLOWN FUSE

The status of up to four fuses on an AC Output, DC Output, or combination I/O module can be made available to your ladder program. Declare the fuses as *inputs* in the software declarations table.

On modules having both inputs and outputs, the points are numbered sequentially (starting at 1 for inputs and starting at 1 for outputs) in the software declarations table as shown in the two examples below.

The 24V DC Output 16 point module with four fuses			The 24V DC I/O 16/8 source module with two fuses		
Name	Type	I/O Point	Name	Type	I/O Point
OUT1	BOOL	O04.01	IN1	BOOL	I05.01
OUT2	BOOL	O04.02	IN2	BOOL	I05.02
.
.
.
OUT16	BOOL	O04.16	IN16	BOOL	I05.16
FB1	BOOL	I04.01	OUT1	BOOL	O05.01
FB2	BOOL	I04.02	OUT2	BOOL	O05.02
.
FB4	BOOL	I04.04	.	.	.
			OUT8	BOOL	O05.08
			FB1	BOOL	I05.17
			FB2	BOOL	I05.18

Remember that the status of a fuse (represented as FB in the table above) is declared as an input.

Fast Inputs

The fast inputs available on the encoder, resolver, and servo encoder hardware modules can be declared as inputs in the software declaration table. The inputs are:

For Channel 1	For Channel 2	For Channel 3	For Channel 4
X.1	X.3	X.5	X.7

where X = the slot number

NOTE: If you are using the FAST_REF, MEASURE, or REGIST functions, you cannot declare the fast inputs in the software declaration table. In that case, use the STATUSV function to find out the status of the fast inputs.

Init. Val. =Init. Val. This field is optional. All variables have a default value equal to zero. A value entered in this field becomes the (new) default value. Default values go into effect when cold restarts are done. They also go into effect for warm restarts for variables that are not retentive.

Initial Values for Bitwise and Numeric Variables

Values can be entered in binary, octal, decimal, or hexadecimal. Binary values must be preceded by a 2#, octal by an 8#, and hexadecimal by a 16#. Decimal numbers are not preceded by any characters.

PiCPro can convert the display of the values from one representation to another:

1. Press <F8> to display a list of the number systems.
2. Move the cursor to the applicable one.
3. Press <Enter> to convert the value.

Examples of the number 45

Binary 2#101101

Decimal 45

Octal 8#55

Hexadecimal 16#2D

Initial Values for Strings

String values are entered in a box, which appears when you start typing.

If you type more characters than you designated for the size of the string, PiCPro changes the size of the string automatically.

Only the first 11 characters of a string display in the Init. Val. field.

```
==Init. Val.==Long
verylongstr
```

To display all characters

Press <ALT-E> when the cursor is in the Init. Val. field

Initial Values for Time Variables

- Values must be preceded by a D# for DATE, TOD# for TIME_OF_DAY or DT# for DATE_AND_TIME.
- PiCPro enters these characters automatically - type only the numbers, dashes, and colons:

Examples

```
D#1991-02-23
TOD#14:43:56
DT#1991-02-23-14:43:56
```

Initial Values for Time Duration Variables

Values are entered in increments of days, hours, minutes, seconds, and milliseconds. You must enter a d, h, m, s, and/or ms after an increment.

PiCPro can convert the display of the values from one representation to another:

1. Press <F7> to display a list of the increments.
2. Move the cursor to the applicable one.
3. Press <Enter> to convert the value.

Examples (maximum values shown)

```
T#49d17h2m47s295ms
T#1193h2m47s295ms
T#71582m47s295ms
T#4294967295ms
```

Long name

```
==Long Name==
long description
```

This field is optional. It is used to enter a longer, more descriptive name than the eight character Name. The Long Name is displayed in this table and in the network using the Longname command. It can be printed (see Print command under Module).

- Enter from 1 to 40 characters.

After the Long Name is entered, only the first 10 characters display. To display all characters, press <ALT-E> when the cursor is in this field.

Declaring Arrays and Structures

This section will cover software declarations of arrays, structures, structures with arrays, an array of structures, and an array of structures with arrays. Each of these represents a method of handling groups of variables.

Arrays

An *array* is a group of variables. Each variable in an array must be of the same data type. Any data type is acceptable. An array can have from 2 to 999 variables and the variables are called *elements*. Arrays are useful for handling large groups of like data items.

As an example, assume that you have a list of 200 rod part numbers. An array could be used to store the list. Each part number is an element in the array. This is illustrated below.

An Array of Part Numbers

Part Number	Data Type
R101	STRING
R102	STRING
.	.
.	.
R199	STRING
R200	STRING

Declaring an Array

To declare an array, follow these steps:

1. Enter the Name and Type for the array variable you want to create.

In the example, RODS is the Name and a STRING with four characters is the data Type.

```
Name=====Type=====  
||RODS      STRING[4]
```

NOTE: If all the elements in your array are to have the same value, enter the value in the Init. Val column now before you create the array. That value will then appear with each element in the array when it is declared. This will not be done for the example since each rod part number is different and will have to be entered individually.

- Position the cursor in any field and press <ALT-A>. The Array Length box appears: Enter a number from 2 to 999 for the array length.

In the example, there will be 200 elements in the array.

Array Length

Enter array length:

Cancel
<Esc>
Accept
<F10>

- Press <F10> to accept your array length.
- After you have entered the number of elements, x, that are to be in the array, the variable gets duplicated that number of times. The index numbers (0...x-1) appear behind the data type of the variable.

In the example, x = 200 and the index numbers will be 0 through 199 as shown below.

Name	Type
RODS	STRING[4](0..199)

- To initialize or change the values of the elements in the array, move the cursor to the Init. Val. field.
- Press <Alt E>.

This box listing all the elements appears. If no initial value was entered before the array was declared, then there will be no value at each element. (Scroll the screen to view all the elements.)

0:	<input style="width: 50px;" type="text"/>
1:	<input style="width: 50px;" type="text"/>
2:	<input style="width: 50px;" type="text"/>
3:	<input style="width: 50px;" type="text"/>
4:	<input style="width: 50px;" type="text"/>
5:	<input style="width: 50px;" type="text"/>
6:	<input style="width: 50px;" type="text"/>
7:	<input style="width: 50px;" type="text"/>
8:	<input style="width: 50px;" type="text"/>
9:	<input style="width: 50px;" type="text"/>
10:	<input style="width: 50px;" type="text"/>
11:	<input style="width: 50px;" type="text"/>
12:	<input style="width: 50px;" type="text"/>
13:	<input style="width: 50px;" type="text"/>

- To enter values, move the cursor with the arrow keys to the desired element. Type in the value you want.

In the example, you would type in R101 at 0, R102 at 1, R103 at 2 through R200 at 199.

- Press <F10>.

The word ..Array.. will show in the Init. Val. field if the initial value for any element in the array has been entered or changed.

Name	Type	I/O Pt.	Init. Val.	Long
RODS	STRING[4[(0..199)		...ARRAY...	

NOTE: If you increase the array length the "added" elements will have the same value as the first element (index 0).

In the example, each additional element beyond 199 would have the value of R101. The value for each additional element can be changed by following the procedure above.

Referencing and Accessing Elements of Arrays in the Module

Array elements are referenced according to the position of the element within the array. The positions are numbered 0 through x-1, where x is the number of elements. The position numbers are called *index* numbers and they act as pointers to the elements. To access an element, you enter the name of the array and the index number of the element.

One of the powerful features of arrays is that the index number can be a variable (USINT or UINT). By using a variable and changing its value, you are able to access every element in the array with a minimal number of commands.

IMPORTANT
<p>Be careful to enter the correct index number for the element you want. PiCPro does not prevent you from entering an invalid index number. If an index number is greater than "x - 1," a memory location after the array will be accessed. The results are unpredictable.</p>

The index number is entered in parentheses after the array name. For the example array, the name RODS followed by constants or a variable called RODNUM can be used to index the array as shown below.

Name with constant or variable	Contents of each element	Data type
RODS(0) or RODS(RODNUM)	R101	string
RODS(1) or RODS(RODNUM)	R102	string
..	.	.
RODS(198) or RODS(RODNUM)	R199	string
RODS(199) or RODS(RODNUM)	R200	string

Structures

A *structure* is a group of variables where the variables can be of any data type except another structure. Each variable that comprises a structure is called a *member*. A structure can have an unlimited number of members.

NOTE
When booleans are used as data types in structures, each boolean occupies a byte of memory and more scan time is required. It is the most significant bit of the byte that indicates the state of the boolean.

When you create a structure (in the software declarations table) you are creating a template or framework for a group of data items.

As an example, assume that you have some information about a rod part that you want grouped together. The data consists of a part number (PART), a description of the part (DESC), its diameter (DIAM), the first operation on the part (OPER), and the time it takes to perform the operation (TIME).

You could create a structure with five members, depicted in the box below.

Member name	PART	DESC	DIAM	OPER	TIME
--------------------	-------------	-------------	-------------	-------------	-------------

Member data

R101	class A rod	126	debur	15m
------	----------------	-----	-------	-----

Data type	string	string	usint	string	time
------------------	---------------	---------------	--------------	---------------	-------------

Declaring Structures

To declare a structure:

1. Enter its name.
 For the example, the name will be ROUTING.
2. Position the cursor on the Type field.
3. Press <F8> to call up the list of types.
4. Select STRUCT as the type.

The screen will display as shown below:

Name_____	Type_____
ROUTING	STRUCT
	END_STRUCT

The END_STRUCT entry is entered automatically by PiCPro.

5. Enter the members for the structure.

For the example, there will be five members entered.

Position the cursor in the Name field on the END_STRUCT line. Press <Insert>. Enter the member name, type, etc. as you would for any variable.

```
Name=====Type=====I/O Pt.====Init. Val.=
ROUTING   STRUCT
.PART     STRING[4]
.DESC     STRING[20]
.DIAM     USINT
.OPER     STRING[10]
.TIME     TIME
END_STRUCT
```

NOTE: A dot or period will be automatically inserted before the member name by PiCPro. All member names (and only member names) start with a dot.

Members can be any data type except another structure.

Referencing and Accessing Members of Structures

To reference a member or access the data, enter the structure and member names, separated by a dot or period.

For the ROUTING structure above with the members PART, DESC, DIAM, OPER, and TIME, the names on the left are used in the module:

Name that you use to reference member	Contents of the member	Data type you assign
ROUTING.PART	R101	string
ROUTING.DESC	class A rod	string
ROUTING.DIAM	126	usint
ROUTING.OPER	deburr	string
ROUTING.TIME	15	time

Structure with Arrays

A *structure with arrays* is a structure where one or more members are arrays of data. Given the definitions and requirements of arrays and structures, it follows that for a structure with arrays:

1. Each element in an array must be the same data type.
2. Each array can be a different data type.
3. The arrays can have different numbers of elements.

As an example, assume that for the structure shown earlier the information for that part is expanded to include data about four diameters, operations, and times. This is depicted in the following diagram of a structure that has three members which are arrays; DIAM, OPER, and TIME.

PART	DESC	DIAM	OPER	TIME
R101	class A rod	126	deburr	15m
		122	polish	30m
		124	plate	90m
		123	polish	30m

Declaring Structures with Arrays

A member of a structure is made into an array the same way that any variable is made into an array. With the cursor in any field on the member line, press <Alt A>. Enter the array length and press <F10>.

Also, initial values are assigned the same way that they are for arrays. With the cursor in the Init. Val. field of the member, press <Alt E>.

DIAM, OPER, and TIME in the example are now arrays with four elements each. Note that the initial value of one or more of their elements has been changed.

```

Name=====Type=====I/O Pt.==Init. Val.=
ROUTING    STRUCT
.PART      STRING[4]
.DESC      STRING[20]
.DIAM      USINT(0..3)          ... ARRAY...
.OPER      STRING[10] 0..3)    ... ARRAY...
.TIME      TIME(0..3)          ... ARRAY...
END_STRUCT

```

Referencing and Accessing Data in a Structure with Arrays

Since an array in a structure is a member, the entire array is accessed as any member of a structure is accessed. Enter the structure name and member name.

The lowest level of data in a structure with arrays is an element in an array. To access or reference an element you use a combination of the methods used to access elements of arrays and members of structures. Input the name of the structure, the name of the member, and an index number for the array element.

For the structure with arrays shown above, where the structure is named ROUTING, array element data is accessed as follows.

DIAM elements	Contents	OPER elements	Contents	TIME elements	Contents
ROUTING.DIAM(0)	126	ROUTING.OPER(0)	deburr	ROUTING.TIME(0)	15
ROUTING.DIAM(1)	122	ROUTING.OPER(1)	polish	ROUTING.TIME(1)	30
ROUTING.DIAM(2)	124	ROUTING.OPER(2)	plate	ROUTING.TIME(2)	90
ROUTING.DIAM(3)	123	ROUTING.OPER(3)	polish	ROUTING.TIME(3)	30

NOTE: A variable instead of a constant can be used for the index number.

Array of Structures

An *array of structures* is an array whose elements are structures. By definition of an array (that each element has the same data type) each structure in an array has the same format. (i.e., if the first member of a structure is a string, then the first member of all the structures in the array is a string, etc.)

As an example, assume that for the array and structure described above, you want to have the structured data for all 200 parts. This is depicted in the following diagram. Data for only 2 of the 200 parts is shown.

R101	class A rod	126	deburr	15m
.			
.			
.			
.			
R200	class B rod	113	deburr	20m

Declaring an Array of Structures

To declare an array of structures, follow these steps :

1. Declare the structure as described above. Add the members (Names, Types, etc.)
2. Position the cursor on the line with the structure Name (ROUTING for the example) in any field.
3. Press <ALT-A>. Enter the array length. (200 for the example.) The length of the array appears by STRUCT.

```

Name=====Type=====I/O Pt.==Init.
ROUTING   STRUCT(0..199)
.PART     STRING[4]
.DESC     STRING[20]
.DIAM     USINT
.OPER     STRING[10]
.TIME     TIME
          END_STRUCT

```

4. To initialize members, position the cursor in the Init. Val field of the desired member. Press <Alt E>.

Three boxes appear as shown below.

Name	Type	I/O Pt.	Init. Val.
ROUTING	STRUCT(0..199)	ROUTING(0)	ROUTING(1)
.PART	STRING[4]		
.DESC	STRING[20]		
.DIAM	USINT		
.OPER	STRING[10]		
.TIME	TIME		
	END_STRUCT		

The cursor is always located in the middle box and only values in this box can be changed. Use the left/right arrow keys to scroll the boxes left and right to access other elements.

Pressing the right arrow key once brings these boxes into view.

Name	Type	I/O Pt.	Init. Val.	
ROUTING	STRUCT(0..199)	ROUTING(0)	ROUTING(1)	ROUTING(2)
.PART	STRING[4]			
.DESC	STRING[20]			
.DIAM	USINT			
.OPER	STRING[10]			
.TIME	TIME			
	END_STRUCT			

And pressing the right arrow key again brings up these boxes.

Name	Type	I/O Pt.	Init. Val.	
ROUTING	STRUCT(0..199)	ROUTING(1)	ROUTING(2)	ROUTING(3)
.PART	STRING[4]			
.DESC	STRING[20]			
.DIAM	USINT			
.OPER	STRING[10]			
.TIME	TIME			
	END_STRUCT			

The number of rows in each box corresponds to the number of members in each structure (all of the structures are alike).

In the example, there are five members. Each box has five rows.

Using the arrow keys, position the desired array element in the middle box.

5. Position the cursor on the line of the member. Enter the value.

Press <Enter> to enter the value.

When all values have been entered, press <F10> to accept them.

For the example the boxes could contain the following:

Name	Type	I/O Pt.	Init. Val.	
ROUTING	STRUCT(0..199)	ROUTING (0)	ROUTING(1)	ROUTING(2)
.PART	STRING[4]	R101	R102	R103
.DESC	STRING[20]	class A rod	class A rod	class B rod
.DIAM	USINT	123	125	113
.OPER	STRING[10]	deburr	deburr	deburr
.TIME	TIME	15m	14m	15m
	END_STRUCT			

IMPORTANT

If the members of a structure are to be initialized in the declarations table, arrays of structures (and arrays of structures with arrays) can have no more than 14 members in the structure if 25 display lines has been chosen in **Computer Setup**, 32 members if 43 lines has been chosen, and 39 members if 50 lines has been chosen.

If the members are not being initialized in the declarations table, the number of members is unlimited.

Since a structure in an array is simply an element in the array, the entire structure is accessed as any element in an array is accessed. Enter the name of the array and the element's index number.

The lowest level of data in an array of structures is a member in one of the structures. To access a member, use a combination of the methods used to access members of structures and elements of arrays. Input the name of the structure, (all structures in an array have the same name), its index number, and the name of the member.

For the array of structures shown above, where the structures are named ROUTING, data is accessed as follows.

To access members in the first structure, enter this	To access members in the second structure, enter this	To access members in the last structure, enter this
ROUTING(0).PART	ROUTING(1).PART	ROUTING(199).PART
ROUTING(0).DESC	ROUTING(1).DESC	ROUTING(199).DESC
ROUTING(0).DIAM	ROUTING(1).DIAM	ROUTING(199).DIAM
ROUTING(0).OPER	ROUTING(1).OPER	ROUTING(199).OPER
ROUTING(0).TIME	ROUTING(1).TIME	ROUTING(199).TIME

NOTE: A variable instead of a constant can be used for the index number.

Array of Structures with Arrays

An *array of structures with arrays* is an array of structures each of which has one or more members which are arrays of data. This is the most complex representation of data that can be created for the PiC900. It combines the four types of groups of data - structures, arrays, arrays of structures, and structures with arrays.

R101	class A rod	126	deburr	15m
•	122	polish	30m
•		124	plate	90m
•		123	polish	30m
•			
R200	class B rod	113	deburr	20m
		110	polish	45m
		112	plate	120m
		111	polish	30m

Building upon the previous examples, assume that structured data is to be created for the 200 parts, and that each structure should contain the arrayed data for four operations. The following diagram depicts the array of structures with arrays. Only 2 of the 200 elements are shown.

Declaring an Array of Structures with Arrays

Declaring an array of structures with arrays is done as explained above by declaring a structure, an array, and a structure with an array.

The initialization of arrays of structures with arrays is the same as described for arrays of structures with an additional step for a member which is an array. That array is initialized like other arrays are initialized.

In the following example, the DIAM member has been declared an array. To initialize the elements in the array, position the cursor in the Init. Val. field on the DIAM line. Press <Alt E> which brings up the three array of structure boxes. Press <Alt E> again to bring up the array box as shown below.

Values are entered as they are for all arrays.

Name	Type	I/O Pt.	Init. Val.
ROUTING	STRUCT(0..199)		ROUTING(0)
.PART	STRING[4]		
.DESC	STRING[20]		
.DIAM	USINT(0..3)	0: 126 1: 122 2: 124 3: 123	
.OPER	STRING[10]		
.TIME	TIME		
	END_STRUCT		

Name	Type	I/O Pt.	Init. Val.
ROUTING	STRUCT(0..199)		ROUTING(1)
.PART	STRING[4]		
.DESC	STRING[20]		
.DIAM	USINT(0..3)		
.OPER	STRING[10]		
.TIME	TIME		
	END_STRUCT		

R102
class B rod
113
debur
15m

Referencing and Accessing the Data in an Array of Structures with Arrays

A structure is accessed the same way that it is accessed for an array of structures.

IMPORTANT

If the members of a structure are to be initialized in the declarations table, arrays of structures (and arrays of structures with arrays) can have no more than 14 members in the structure if 25 display lines has been chosen in Computer Setup, 32 members if 43 lines has been chosen, and 39 members if 50 lines has been chosen.

If the members are not being initialized in the declarations table, the number of members is unlimited.

An array is accessed the same way that a member in an array of structures is accessed.

The lowest level of data in an array of structures with arrays is an element in an array. To access an element, you use a combination of the methods used to access members and arrays. Input the name of the structure, its index number, the name of the member, and the index number of the element.

For the array of structures with arrays shown above, where the name of each structure is ROUTING, the diameter data is accessed as follows for part #s R101 and R200.

To access diameter data for part # R101, enter this:	Contents	To access operation data for part # R200, enter this:	Contents
ROUTING(0).DIAM(0)	126	ROUTING(199).OPER(0)	deburr
ROUTING(0).DIAM(1)	122	ROUTING(199).OPER(1)	polish
ROUTING(0).DIAM(2)	124	ROUTING(199).OPER(2)	plate
ROUTING(0).DIAM(3)	123	ROUTING(199).OPER(3)	polish

Module Menu - Network

The commands on the Network, Element, and View menus are covered in the next three sections.

Network

- Insert <Ins> Inserts a network where the cursor is located, and renumbers the networks if the cursor is not at the end of the module. Pressing the <INS> key has the same effect as selecting this command.
- Delete Deletes the network the cursor is located on. PiCPro will prompt you first. Pressing the key has the same effect as selecting this command.
- Edit <Enter> Enters the network editing mode allowing you to make changes to an existing network.
- Pressing <Enter> when the cursor is on the network number has the same effect as selecting this command.

On-Line Editing

In normal program development, you change your ladder and then download the entire module to the PiC. The scan is stopped during this download and must be started again when the download is complete.

There may be times when you need to edit the ladder on-line without stopping the scan. If you have made changes to a ladder that is currently being scanned in the PiC and do a **Module Download**, the following box appears. You can choose to download the entire module stopping the scan (off-line edit) or to send a "patch" with the scan running (on-line edit).

```
Stop the scan and download the entire module,
or allow the scan to continue and download only
the changes as patches?

Stop the scan and download
Patch the scanning ladder
```

When you choose *Patch the scanning ladder*, only the changes you have just made will be downloaded and the scan is not stopped. When the patch download is complete, the following box appears and tells you what resources you still have available.

```
* Resources Still Available *
      8174 data bits
      32758 data bytes
      260656 code bytes
          99 patches
      26 label/function links
Use <Ctrl-S> to abort the latest changes
Press any key to continue
```

Some things to keep in mind when using the on-line edit (patching) feature are listed below.

1. The application program cannot be in EPROM memory.
2. A downloaded patch becomes effective at the beginning of the next scan.
3. You can do up to 100 patches in a module with a limit of 40 internal tasks in any one patch download, depending on memory available. If you exceed 40 tasks at one time, you must do a full download stopping the scan.

Every time a network is modified, it is considered one patch but could include several tasks.

The 'Resources Still Available' box lists the number of patches you have left.

4. PiCPro establishes a link table for labels and/or function/blocks added during on-line editing.

The 'Resources Still Available' box lists the number of links you have left.

5. You can use <Ctrl>S to undo the last patch(es) downloaded to the PiC. Although <Ctrl>S removes the patch(es) from the PiC, it does not remove them from the LDO on your workstation. You can now redo the patch(es) in the LDO on your workstation and do another patch download.

NOTE: <Ctrl> S removes the operation but not necessarily the effect of that operation from the PiC. For example, if you add a new network that turns on a new coil and do a patch download followed by a <Ctrl> S, the network is removed from the PiC but the coil will remain on.

6. The changes you make and download with on-line edit are a permanent part of the program in the PiC. They are included in any backup from the PiC.
7. When working with UDFBs, on-line editing can be done in the main module and in the source ladders of any UDFBs.
8. If you have ignored a Save request from PiCPro at some point, on-line editing is prohibited.
9. You can add new entries to the declaration table anywhere in the table, but the physical address will be at the end of the list in the PiC.
10. Changes to existing hardware or software declarations will prevent a patch download.
You must do a full download to incorporate them into the module in the PiC.
11. You can add new variables with initial values with on-line edit, but you cannot change or add initial values to existing variables without doing a full download.

Find Locates (and scrolls to) a network. When you select this command you are prompted to enter the number or label of the network you want to find. Pressing <F8> will bring up a list of any network labels used in the ladder.

Mark (start) Moves, copies, or deletes one or more networks. This command applies only to whole networks (not portions), and moves or copies them to a location in front of another network (not within one).

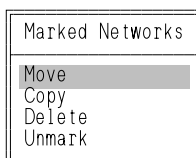
The steps are:

1. Locate the cursor on a network
2. Select the Mark (start) command.

The network gets marked (highlighted) and the menu changes to Mark (end). To mark more than one network, scroll the screen. Do not move the cursor if you want only one network marked

3. Select Mark (end).

The screen displays the following prompt:



4. Select Move, Copy, Delete or Unmark. The menu changes to Mark(quit). You can select "quit" to unmark the network and not make any changes to it.

- Move** The screen displays the following prompt at the end of the marked selection.
- Move here <Enter>, or quit <Esc>
- Scroll the screen to the location where you want the marked network(s) to be moved. The prompt moves too. Press <ESC> to exit - unmark the network and not move it. Press <Enter> to move the network. It gets deleted from its previous location and inserted in front of the network that is highlighted by the prompt.
- Copy** Works identically to the Move command, except the network is copied. The original network remains in place and a copy of it is placed at the location of the prompt.
- Delete** Works identically to the Delete command and DEL key (described above). —
- Unmark** Unmarks a network, and exits out of the marking procedure.

- Label** Assigns a label to a network. This is the label that must be present for a Jump command. Position the cursor on the network number before selecting this command. Labels are 1 to 8 characters - alpha, numeric, or underscore (_). The first character cannot be numeric.
- If a Jump command is in the network and there is no network with the corresponding label, an error message appears when you try to download the module.
- Comment** Calls up the "comment editor" which can be used to insert comments (documentation) into a network. Comments make up the .REM file. They are entered and get displayed directly below the network number. They can be up to 100 lines in length, with 80 characters per line. Only the top line of a comment is displayed at any time except when this command is selected. A down arrowhead at the end of a comment line indicates that there is more than one line of comments. Refer to the <F9> Help screen, after selecting the Comment command, for detailed information on comment editor commands.

Module Menu - Element

Find You can select the type of find you want to do from the following prompt which is displayed when the Find command is selected:

Find
All other types Coils only Function/block type Jump to label/sub.

The search for any type is done either forward or backwards from the cursor location.

After the first occurrence is found, the following box appears and you can choose *Next* to search forward or *Previous* to search backwards for another occurrence.

Find
Next Previous

All other types

This selection allows you to find any variable included in your program.

Find Variable	
Search variable:	
Dir. forward: Yes	
Cancel <Esc>	Accept <F10>

Enter the name of the variable. NOTE: Pressing <F8> will bring up a list of variables.

Leave the default direction forward or press N to enter No and search backward. Press <F10> to begin search.

Coils only

This selection allows you to find any coils included in your program.

```
— Find Coils Only —
Search coil:
Dir. forward: Yes

Cancel      Accept
<Esc>      <F10>
```

Enter the name of the coil. NOTE: Pressing <F8> will bring up a list of available coils.

Leave the default direction forward or press N to enter No and search backward. Press <F10> to begin search.

Function/block type

This selection allows you to find any function/block included in your program.

```
— Find Function Type —
Search type:
Dir. forward: Yes

Cancel      Accept
<Esc>      <F10>
```

Enter the name of the function/block. NOTE: Pressing <F8> will bring up a list of available function/blocks.

Leave the default direction forward or press N to enter No and search backward. Press <F10> to begin search.

Jump to label/sub

This selection allows you to find any jump to a label/subroutine command included in your program.

```
— Find Jump Label/Sub. —
Search label:
Dir. forward: Yes

Cancel      Accept
<Esc>      <F10>
```

Enter the label. NOTE: Pressing <F8> will bring up a list of available labels.

Leave the default direction forward or press N to enter No and search backward. Press <F10> to begin search.

Replace Replaces a variable or function block with a different variable or function block which may or may not be of the same type. This command is applied forward from the cursor location. If there is ladder logic before the cursor it is not searched. The substituting element must have been declared or this command will not execute.

When the replace command is selected the following prompt appears:

After the name of the element has been entered, PiCPro searches for the first occurrence of the variable/function block. When the first one is found, this prompt is displayed:

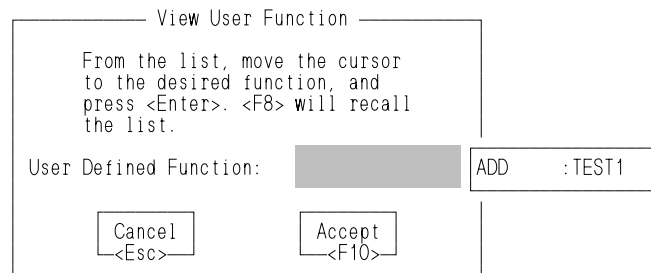
If you respond "yes", the first change is made and the second occurrence (if it exists) is displayed. The process continues until all occurrences have been located.

If you respond "no", the first occurrence is not changed and the second occurrence is searched for and displayed (if it exists). The process continues until all occurrences have been located (or you press ESC).

If you respond "global", all occurrences (after the cursor) are changed.

Module Menu - View

- Zoom** Toggles the view of the module between the normal display and an expanded display showing any longnames assigned to variables.
- User Function** Opens the module that was used to create a user defined function block (UDFB) when running a module containing the UDFB. Allows you to view through animation the logic occurring within the function block.
- Prev. Function** Toggles back to the user defined function block that was opened previously to the one in view now. Allows you to view through animation the logic occurring within the function block.

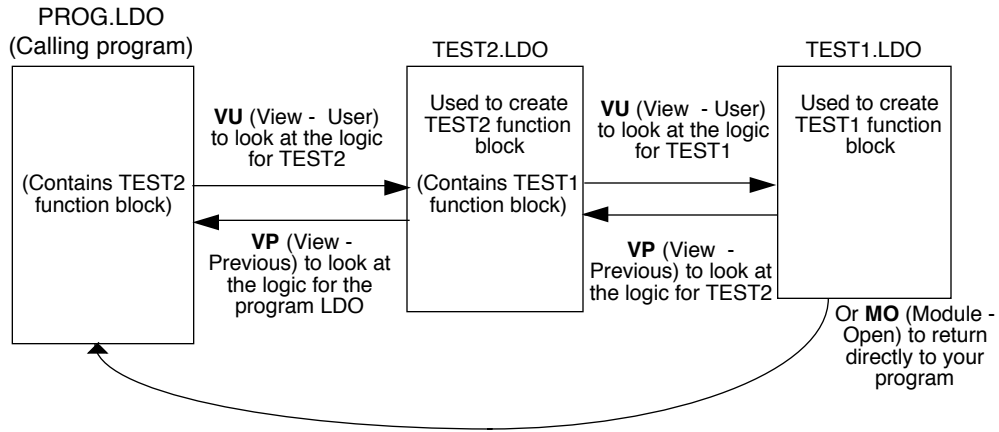


NOTE

The View User and View Previous Function features are only available from within a calling program. An example of how they can be used is shown below.

Module TEST1.LDO is used to create the user defined function block TEST1. The function block TEST1 is used in the logic for module TEST2.LDO to create TEST2 function block. TEST2 is used in your final module which is your calling program.

With the View - User Function feature, you can view and animate each module in descending order. With PROG.LDO running you can view TEST2.LDO. From TEST2.LDO you can view TEST1.LDO as illustrated below. With the View Previous Function, you can step back through the viewing order.



NOTES

CHAPTER 4 System Integration

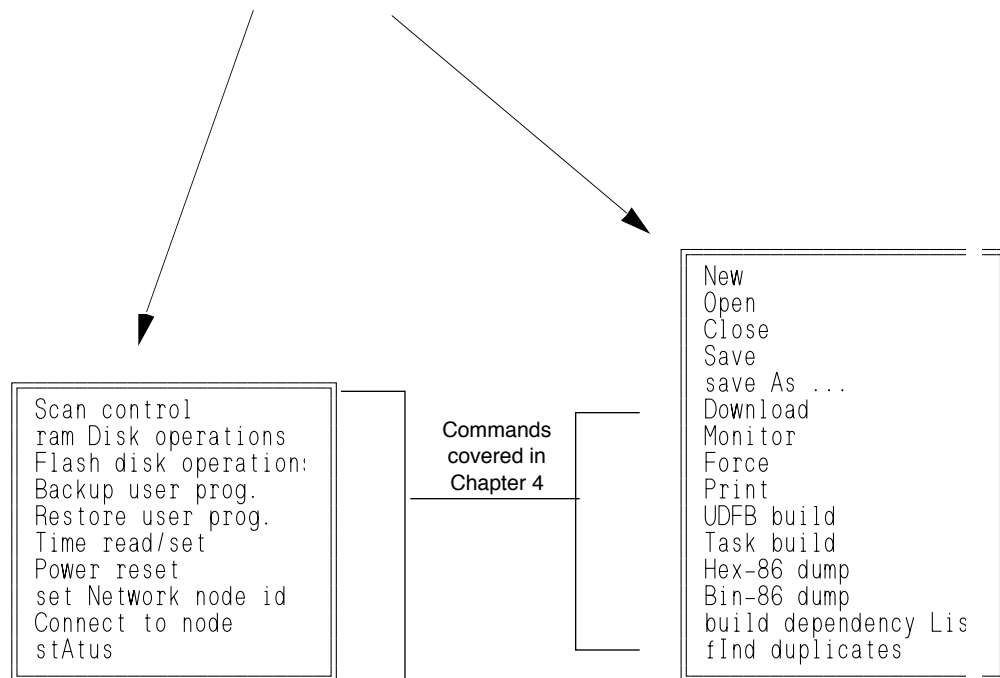
Introduction

Once an application program has been created with PiCPro or PiCServoPro software, there are several PiCPro commands that allow you to integrate the program with the PiC900.

The commands are found under the Processor menu and under the Module menu as shown in Figure 4-1 and will be covered in this chapter. The Module menu commands will be covered first. Note that the last 10 Module menu commands are covered here.

FIGURE 4 - 1. PiCPro commands covered in Chapter 4

Workstation Processor Module Declarations Network Element View



Module Menu - Download

When you create a New module or Open an existing one, the Download command appears on the expanded Module menu. (See Figure 4-1.)

```
Download complete:
* Memory Usage *
86 of 8k data bits
268 of 32k data bytes
308 ladder code bytes
704 total code bytes
Press any key to continue
AA146-1190
```

MD (Module - Download) These hot keys tell PiCPro to send the module on screen to the PiC900 memory. A message in the lower left corner says "Downloading network - - " with a number that increases from 1 to the total number of networks in the module.

When the download is complete, the application memory usage box appears. The memory areas are as follows:

8 K data bits	for	Booleans
32K data bytes	for	Variables
ladder code bytes	for	Your application ladder
total code bytes*	for	Your application ladder plus libraries

*The total code bytes must not exceed the application memory (64K or 256K) available on your CPU module.

The specific memory areas cannot accept overflow from another area. If, for instance, the value in the "data bits" category is larger than 8 K, overflow will occur.

When the program is downloaded, anything that was in PiC900 memory is automatically erased and the scan stops.

After an application program has been downloaded into the PiC900, you can continue to edit the program in the workstation using PiCPro commands. However, any changes do not take effect until the program is downloaded again.

Downloading an application program compiles the ladder to an executable program in the PiC900. The source form of your application program is still on your workstation disk. It is the source form that allows you to make changes, run animation, etc.

Although you can use the Backup and Restore commands under the Processor menu in PiCPro to back up your executable application program, this does not backup your application in its source form.

Make backup copies of all the files listed below to ensure that you will always have a copy of your application program in its source form.

The filenames with the following extensions must be backed up. The two files in the column on the left must be backed up when using PiCPro and PiCServoPro. If you are using PiCServoPro to create the application program, the additional three files on the right must also be backed up.

PiCPro/PiCServoPro	PiCServoPro
.LDO	.SRV
.REM	.PRO
	.LIB

NOTE: If you have more than one of the files listed on the right that are part of this application, be sure to back up all of them.

The definitions of these extensions are found in Appendix A.

Troubleshooting "Download"

If PiCPro cannot communicate with the PiC900 system, this message comes up on screen.

```
Error :Control Communications Not Connected
Press any key to continue, <F9> for help!
```

AA147-1190

These steps can help isolate the problem.

1. Check that the computer end of the cable is plugged into the computer serial port, and the PiC900 end into CPU Port 1. Make sure the connections are tight.
2. Check that the CSM power LED is on.
3. Type **WC** (Workstation - Computer) to check if the correct computer serial port is specified. If you made changes in the Computer setup screen, the new setup parameters can not be used until you reload PiCServoPro.
4. Turn off the Run key and the CSM power switch, then turn the CSM switch back on. Try again to download.
5. If the module still doesn't download, check each of these possible problems:
 - an incorrect or damaged cable.
 - a malfunctioning serial port on the workstation.
 - a malfunctioning serial port on the CPU module.

PiCPro has a number of error-trapping routines that operate while you are creating an application program. However some potential problems cannot be checked until the program is being downloaded. They are pinpointed so you can correct them easily.

```
      Data typing error!
      TIME required, NUMERIC CONSTANT supplied
      Network:5 row:3 col:1

      Download suspended!

      Press any key to continue
```

```
      TEST1 calls for a board in slot 4,
      which is configured as an empty slot.

      Press any key to continue
```

AA148-1190

```
Error :Control Communication Command Not Executable

      Press any key to continue, <F9> for help!
```

```
Error :Incompatible Variable 'X'
Use of 'LREAL' type requires a CPU with an NPX processor.
      Do you wish to continue?
      Yes
      No
      Ignore All

      Press any key to continue
```

Inputs and outputs of Functions/Function Blocks are cross-checked during Download to see if they are the correct data types.

If this message appears, correct the error in data type and Download the application program again. If there should be another data type error, it will be caught on the second Download.

Each physical input or output assigned to an element in the Software Declaration table is compared with the Hardware Declaration table to make sure all necessary hardware modules were declared.

If this message appears, correct the error in either the Software or Hardware Declaration table.

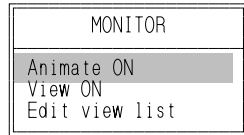
This message might appear if you start the PiC900 using the Scan control commands. It means that the Run key on the CSM is not turned on.

An error like the one on the left will appear if your ladder contains data that requires a math coprocessor installed on the CPU to process and there is none.

NOTE: If you choose to continue and need a math coprocessor, you will get a scan loss error when you attempt to scan the ladder.

Module Menu - Monitor

The Monitor options instruct the PiC900 to send the application status data to the computer workstation and display it on the screen.



AA150-1190

MMA (Module - Monitor - Animate ON) Toggles Animate on and off.

MMV (Module - Monitor - View ON) Toggles the View ON on and off. When it is on, variables entered in the View Variables List appear in the lower right of the display, with their current state or value.

MME (Module - Monitor - Edit view list) Brings up the View Variable List. Any variables you want to monitor continually can be added to the list. The list is Saved in its own file (with extension .RTD for Real Time Display) whenever you Save the application program.

These three commands will be covered next.

Monitor Menu - Animate ON

With animate on, an image of the flow of power in the system is visible on the workstation screen. If the RUN key in the CSM is turned on, the energy flow on screen changes in response to program commands. If it is turned off, the image is static reflecting the state of the application. See Figure 4-2.

Contacts, coils and connectors that are energized are highlighted on screen, and those that are not energized are not highlighted. You can watch the flow of power through all visible networks.

Numeric variables have their contents displayed above the variable name, and they change as the contents change. For instance, you can watch a timer increment to its preset value, time out, and start again from 0 ms.

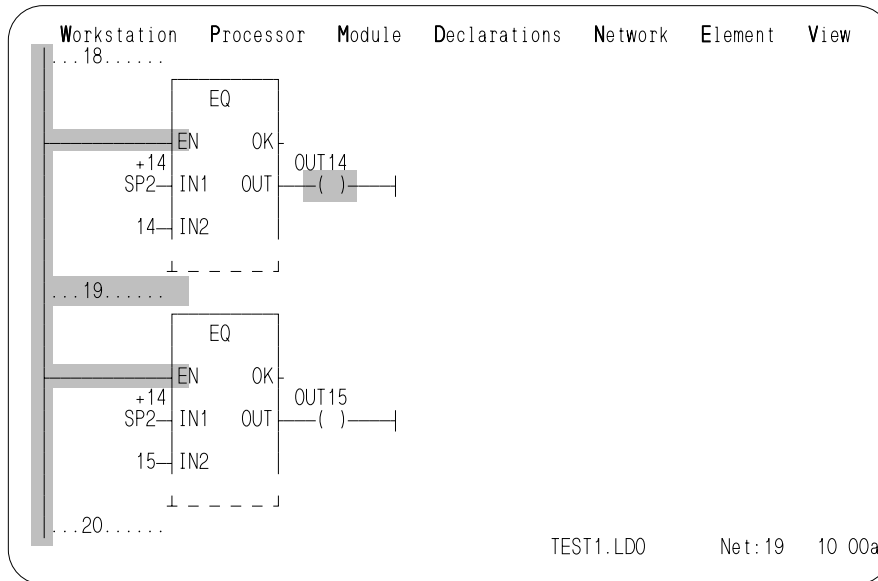
String variables display only the first 10 characters, enough to identify them. To see the whole string you must add it to the View Variables List and turn View on.

NOTE

A message telling you that the monitor functions are inhibited appears on the screen if you attempt to monitor an LDO that has been changed since the last download or has never been downloaded.

The message tells you the name of the LDO currently in the PiC900.

FIGURE 4 - 2. Display in Animate Mode



In this example, both EQ functions are always energized from the left power rail. Each one compares a different constant with the same variable. Earlier in the program, a timer and an ADD function set the integer variable SP2 to increment every 0.5 second. At the moment SP2 has reached 14, so OUT14 is energized and OUT15 is not. In half a second, OUT14 will be de-energized and OUT15 energized.

Scroll the display up/down or pan left/right to see (portions of) other Animated networks.

If you type **NE**, **NI**, or **Enter** to enter a network, the Animate highlighting turns off.

If you edit a network, Animate will not turn on when you exit from the network. A message tells you that the workstation version is no longer identical with the one in the PiC900. Download the module, then type **MMA** again to turn Animate back on.

A complex element type consisting of a Boolean array with a variable array index cannot be evaluated by animation. (It is still evaluated by the PiC900 ladder.) The element (contact) on screen blinks when Animate is turned on. The power flow display stops at this element. Elements to the right of the blinking contact will have their on/off status displayed.

When working with floating point numbers, animation will return the following labels if the result is infinite, indefinite, or not-a-number. The OK on the function will not be set.

Value	Output
+ infinity	1.#INF
- infinity	-1.#INF
Indefinite	<i>digit</i> .#IND
Not-a-number	<i>digit</i> .#NAN

Monitor Menu - View ON

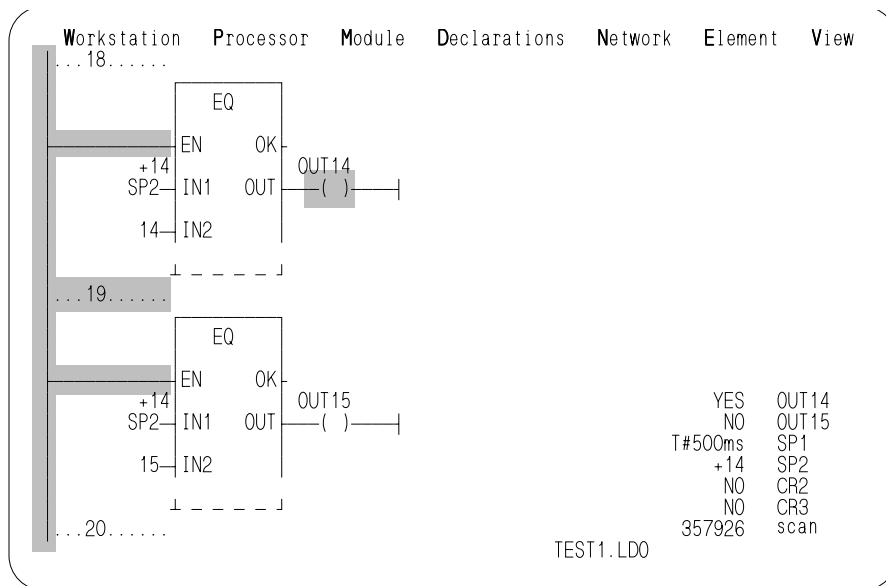
With View ON turned on, the list of enabled variables appears in the lower right corner of the screen, each with its current value. (See Figure 4-3.) If the application is running, these values change constantly to reflect the corresponding values in PiC900 memory.

The Variables display remains in the same place on screen as you toggle Animate on and off, use either Find option, or use the arrow keys to scroll the ladder. Type **MMV** again to turn it off.

The Variables display always contains scan. Its value starts at 0 when the PiC900 starts scanning, and increments each scan as long as the PiC900 is in RUN mode. Toggle the Run key on the CSM to reset the count to 0.

Animate and View options complement each other. (See Figure 4-3.) You can monitor one network and keep track of the values of variables elsewhere in the program at the same time.

FIGURE 4 - 3. Animate and View Options Both Turned On.



Error : Control Communication
Press any key to continue, <F9> for help!

You may get this message when you try to access Animate or View. First check that the PWR LED on the CSM is on. If the LED is off, the PiC900 system doesn't have power, and it cannot send data to the workstation.

If the PiC900 system has power, check the communication cable and try again.

Monitor Menu - Edit View List

```
Enb-----View Variables List-----
OP14 -> Boolean
OP15 -> Boolean
* OUT14 -> Boolean
* OUT15 -> Boolean
* SP1 -> Time (Milliseconds)
* SP2 -> Signed decimal
  CR1 -> Boolean
* CR2 -> Boolean
* CR3 -> Boolean
* scan
Alt-E toggles Enb-----Ins (before) or Del
```

MME (Module - Monitor - Edit view list) brings up the View Variables list. Press <Ins> and press <F8> to bring up a list of variable types. Choose the type from this list and press <Enter>. A list of that type of variable in the LDO will appear. Select the one you want to add to the view list and press <Enter>. If the variable is an array, it will be inserted into the list as *ARRAY()*. You must enter the index in the parenthesis.

Or you can type the name of the first variable you want to monitor. If the name is not entered in the Software Declaration table, a pop-up tells you Symbol not found.

When you press <Enter> a selection list gives all

If the variable is a contact or coil its data type is always a Boolean, so when you press <Enter> "Boolean" is entered.

If the variable is a string, its data type is always a string, so when you press <Enter> "String" is entered.

Press <Enter> or <F10> to Accept the table and return to the main screen.

The view list holds lines of variables equal to the number of lines on your screen minus three. For example, if your screen has 25 lines, you can enter 22 lines of variables. If the list is full and you need to view other variables, you must delete some entries to make room. Press the key and a pop-up asks you to confirm that you want to Delete the highlighted entry.

NOTE: The scan entry cannot be deleted or disabled.

The Enb (enable) option lets you display some variables from the list and suppress others. A variable preceded by an asterisk is Enabled. Highlight an entry in the View Variables list and press <Alt>E to toggle Enb off; press <Alt>E again to toggle it on. Only Enabled variables are displayed when the View Variables option is turned on. You can eliminate distractions by displaying only variables you are concerned with at the moment.

NOTE: If there is not enough memory to store the monitored variables, an insufficient memory message will be displayed on the bottom line.

Module Menu - Force

Force is a way to alter the contents of up to 64 variables while the program is running. It is a powerful tool in isolating a problem.

In testing (debugging) a software module you can Force a value into

- an input element to simulate the data input from an application device which has not been connected.
- an element that enables a function or function block you want to check.
- a timer's preset variable to check what happens when it times out.
- In troubleshooting an application you can Force a value into an output element to locate a problem when troubleshooting an application.
- selected elements between an input that works and an output that doesn't, to isolate an internal problem

WARNING

Do not Force any variable in a program that is running an application until you are sure you understand all effects the changes may cause.

A malfunctioning program may:

- cause injury
- damage the machinery

Variables are forced by entering their names and values in a list, by marking them (in the list) to be forced, and by turning on the forcing option.

An example of a forcing list is shown below.

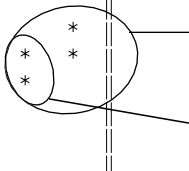
```
Name-----Value-----Enb=Grp
OUT13      0
OP13       0
OUT14      1
OUT15      0
OUT16      0
OUT17      ?not found?
end_table
└─ forcing is off= group is off
```

In the example list above, OUT14 and OUT15 are "marked" for forcing. This is designated by the asterisks in the Enb (enable) column. OUT14 and OUT15 will equal 1 and 0, respectively, when forcing is turned on.

OP13 and OUT14 are marked for "group" forcing. This is designated by the asterisks in the Grp (group) column. OP13 and OUT14 will equal 0 and 1, respectively, when forcing and group are turned on.

NOTE: Group forcing is on only when "forcing" is on. When group forcing is on, all variables with an asterisk in either the Enb or Grp columns are forced. When forcing is on and group forcing is not on, only variables with an asterisk in the Enb column are forced.

Name	Value	Enb	Grp
OUT13	0		
OP13	0	*	
OUT14	1		*
OUT15	0	*	
OUT16	0		
OUT17	?not found?		
end_table			



IMPORTANT

Animation may not reflect forced conditions. Physical inputs are forced at the beginning of the scan. All other variables are forced at the end of the scan. The writing of variable values by the LDO is not inhibited by forcing. Therefore it is possible to change the state of a control relay or coil during the scan, but not "see" the change in animation.

When Force is selected from under the Module menu, this screen appears:

Force

Forcing <Ctrl-F> OFF
 Group enable <Ctrl-G> OFF
 Edit force list

The three options on this screen are:

- | | |
|---------------------------|---|
| Forcing <Ctrl-F> OFF | <p>You can toggle the forcing feature on and off by performing any of the following:</p> <ol style="list-style-type: none"> 1. press <Ctrl-F> 2. use hot keys (M, F, F) 3. position the cursor on this option and press <ENTER> |
| Group enable <Ctrl-G> OFF | <p>You can toggle group forcing on and off by performing any of the following:</p> <ol style="list-style-type: none"> 1. press <Ctrl-G> 2. use hot keys (M, F, G) 3. position the cursor on this option and press <ENTER> |
| Edit force list | <p>You can call up the forcing table by performing any of the following:</p> <ol style="list-style-type: none"> 1. use hot keys (M, F, E) 2. position the cursor on this option and press <ENTER> |

To enter variables and values in the table:

1. Position the cursor (on the line) following the line where the variable should appear and press <Insert>.
2. Press <F8> to bring up a list of data types. Choose the type and press <ENTER> This brings up a list of variables of that type in the LDO. Highlight the one you want to enter on the Force list and press <ENTER>. If the variable is an array, it will be inserted into the list as *ARRAY()*. You must enter the index number in the parenthesis.
You can also type in the variable name.
Use <tab>, <ENTER>, or right arrow to move the cursor to the value field.
3. Enter a value.
Use <tab>, <ENTER>, or right arrow to move the cursor to the Enb field.
4. Press any key, except <Delete> or spacebar, to turn enable on for the variable. (An asterisk appears when the variable is enabled.)
Use <tab>, <ENTER>, or right arrow to move the cursor to the Grp field.
5. Press any key, except <Delete> or spacebar, to include the variable in a group. (An asterisk appears when the variable is in a group.)

To delete a variable from the table:

Position the cursor on the variable name and press .

To disable or "ungroup" a variable:

Position the cursor in the Enb or Grp field and press <Delete> or the spacebar.

To change table data:

Position the cursor on the name or value and type over the data.

If you type in the name of a variable in the Force list that has not been declared, a message appears in the Value column that says ?not found?.

If a value that is out of range is entered for a variable, the value reverts to its former value.

Module Menu - Print

PiCPro can Print to a printer, to a modem, or to a file on disk. A Printed file on disk contains codes for carriage returns and other printer requirements.

If you Print to a file on disk, the primary name is the same as the one you loaded in PiCPro, but the extension is .LST instead of .LDO. In creating this text file, PiCPro inserts carriage returns, form feeds, and so on. You can use DOS commands to send this file to the printer as needed.

```
Print Module
Print to: TEST1.LST
Cross reference: Yes
Declarations <F8>: Order entered
Networks: Yes
From network: 1
To network: 1
Full comments: Yes
Long names: Yes
Cancel <Esc> Accept <F10>
```

MP (Module - Print) brings up a dialog box.

Print to: The default entry sends the program to a disk file. If you want a hard copy of the ladder diagram, change the first option to the port where the printer cable is connected. Common port names are LPT1 (parallel port 1) or PRN (a printer port).

Cross reference: Leave the default Yes to include in the printout a list of all the elements in the program, with the numbers of all the networks each one was used in.

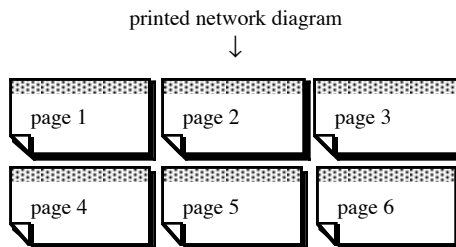
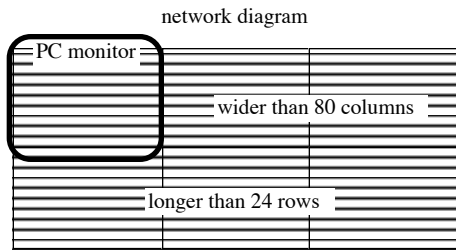
```
Print Module
Print to: TEST1.LST
Cross reference: Yes
Declarations <F8>: Order entered
Networks: Yes
From network: 1
To network: 1
Full comments: Yes
Long names: Yes
Cancel <Esc> Accept <F10>
```

```
Print Options
Order entered
Alphabetically
By type
None
```

Declarations: The software and hardware declaration tables can be printed. Press <F8> to bring up the list of Print Options shown on the left. The first three choices apply to the software declarations table.

The default is to print the software declarations table in the order the variables were entered. It can also be printed listing the variables alphabetically or listing the variables by data type. You can also choose to not have the declarations printed by selecting None.

If None is chosen, neither the software nor the hardware declarations will be printed.



Networks: Leave the default Yes to include in the printout a list of all the elements in each network.

From/To network: You can Print part of a module. The default gives the first and last network numbers in the module, but you can choose to begin or end with any other networks.

Full comments: Leave the default Yes to print all lines of comments that are entered in the application program. No will print only the first line of comments on each network.

NOTE: To force a page eject, type \$P where you want the top of the page to appear.

To have a heading appear on the top of each page in your printout, type \$H (left justified), followed by the heading you want when entering comments. This heading will continue to appear on each page until the next \$H occurs.

Long names: Leave the default Yes to print the long name (as declared in the software Declarations table) above the name of all variables. No will print only the Name.

Print out characteristics

When you print out a copy of your LDO, the symbols described below may appear.

Symbol	Description
&	Appears in the cross reference section of the printout and also in <i>Network Coil Usage</i> message at the bottom of a network telling you that the output/coil is sourced in more than one place.
/	Appears in the cross reference section of the printout and also in <i>Network Coil Usage</i> message at the bottom of a network telling you that the contact is normally closed.
@	If any variable name in the LDO network is more than eight characters, an alias is substituted at the location. (@001 would represent the first substitution, @002 the second, and so on.) The <i>Aliases used in above network</i> message at the bottom of the network lists all the aliases used in that network and the variable names they replaced.
I04.01 or I2.04.01	Appears under a real world input to indicate the following: I(I)np ^u t in master rack , slot 4 (04), channel 1 (01) or I(I)np ^u t in expansion rack 2 (2), slot 4 (04), channel (01)
O03.01 or O2.03.01	Appears under a real world output to indicate the following: O(O)ut ^u t in master rack , slot 3 (03), channel 1 (01) or O(O)ut ^u t in expansion rack 2 (2), slot 3 (03), channel (01)

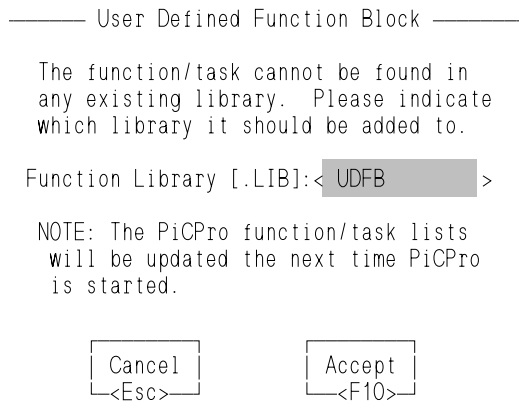
Troubleshooting the Print Option

PiCPro sends out a file, but cannot check if it reaches the printer or modem.

- If the printer operates but the printout doesn't look like the ladder diagram on screen, you may need to change the printer settings. This is discussed in detail in the section on Printer Setup in Chapter 1.
- If the file does not print, these steps can help isolate the problem.
 1. The printer must be on-line and supplied with paper.
 2. Check that the computer end of the cable is plugged into the correct port. Make sure the connections are tight at both ends.
 3. Run the printer's diagnostics tests, then try again to print the program.
 4. If the printer still does not print, check each of these possibilities:
 - an incorrect or damaged cable.
 - a malfunctioning port on the workstation.
 - a malfunctioning port on the printer.

Module Menu - UDFB Build (User defined function block)

The User defined function block command allows you to convert the logic in the module (.LDO) you have created into a single function block.



MU (Module - UDFB build) brings up the dialog box on the left.

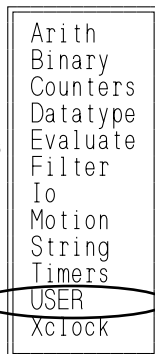
Type in the name you want to appear in the **USER** library list of the Functions menu. It will receive the .LIB extension automatically.

The new library containing your function block can be accessed from the Functions menu when PiCPro is restarted. See below.

IMPORTANT

Always keep the UDFB .LIB file and the corresponding UDFB .LDO file in the same directory.

USER will show up in the Functions menu after you have made a UDFB or a TASK and restarted PiCPro. Accessing USER will up any libraries you have created to hold your UDFBs or TASKs.



When PiCPro is restarted, the Functions menu will contain the word **USER**.

FU (Functions - USER) accesses a list of all libraries you have created for UDFBs.

Accessing the library will bring up a list of individual function blocks you have created and stored in that library.

The UDFB can now be used in any LDO you create. It can also be included as part of an LDO that you want to use to create another UDFB.

See Chapter 8 for more information on UDFBS.

Module Menu - Task Build

The task build command allows you to convert the logic in a task module (.LDO) you have created into a TASK function block that can be used in your main LDO

IMPORTANT

Always keep the TASK .LIB file and the corresponding TASK .LDO file in the same directory.

———— User Defined Task ————

The function/task cannot be found in any existing library. Please indicate which library it should be added to.

Function/task Library [LIB]:< >

NOTE: The PiCPro function/task lists will be updated the next time PiCPro is started.

Cancel <Esc> Accept <F10>

MT(Module - Task build) brings up the dialog box on the left.

Type in the name you want to appear in the **USER** library list of the Functions menu. It will receive the .LIB extension automatically.

The new library containing your task can be accessed from the Functions menu when PiCPro is restarted. See below.

USER will show up in the Functions menu after you have made a UDFB or a TASK and restarted PiCPro. Accessing USER will bring up any libraries you have created to hold your UDFBs and/or TASKs.

Arith
Binary
Counters
Datatype
Evaluate
Filter
Io
Motion
String
Timers
USER
Xclock

When PiCPro is restarted, the Functions menu will contain the word **USER**.

FU (Functions - **USER**) accesses a list of all libraries you have created for tasks and/or UDFBs. Accessing the library will bring up a list of individual tasks you have created and stored in that library.

The task can now be used in an LDO you create.

See Chapter 9 for more information on TASKs.

Module Menu - Hex-86 Dump

MH (Module - Hex 86 dump) An .LDO file that has been Opened in the computer workstation can be saved in Intel 8086 Hex format using the Hex-86 dump command on the Module menu. The name automatically has the same filename as the .LDO file with the .HEX extension. This file may then be copied to an EPROM programmer to program your EPROMs.

The CPU write-ups in the Hardware Manual contains information on EPROM programmers and on the types of EPROMs to use.

Module Menu - Bin-86 Dump

MB (Module - Bin 86 dump) An .LDO file that has been Opened in the computer workstation can be saved as a binary file using the Bin-86 dump command on the Module menu. The name automatically has the same filename as the .LDO file with the .BIN extension. Thus an application program can be stored on one disk or directory in both forms.

This .BIN file can then be loaded into the PiC900 using the Restore command on the Processor menu.

Module Menu - Build Dependency List

ML(Module - build dependency List) This command allows you to build a complete list of all the files that are related to the module you are currently running in PiCPro. The list can include all the LDO source, REM source, UDFB/TASK sources, and UDFB/TASK libraries needed to run the program.

If you have Flash memory installed on your control, the dependency list can be used to send all the listed files to the FMDSK using Flash Disk operations under the Processor menu.

```
Build Dependency List
Output File Name:<          >
Select which of the following to include:
Program Source: Yes
UDFB/TASK Source: Yes
UDBF/TASK Libraries: Yes
List of Other files:<          >
Cancel  Accept
<Esc>  <F10>
```

ML (Module - build dependency List) A file containing a list of all the files the module currently running in PiCPro depends upon can be built with this command.

Enter a name for the output file at **Output File Name**.

The default for the next three items is to include all of them.

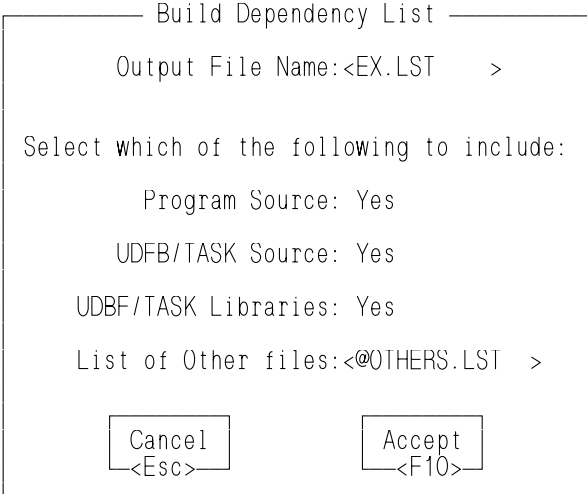
The **List of Other files** allows you to include additional files in the dependency list. If the file you enter here is a list of several files, use the @ in front of the filename to include all the connected files in the dependency list.

Press F10 to build the dependency list.

To view the dependency list, exit PiCPro and open the output file in any text editor. An example of this text file follows.

Example

In this example, an output file called EX.LST was created with the build dependency list command. In the **List of Other files** box, a previously created list of other files called OTHERS.LST was entered in the format shown. The @ indicates that OTHERS.LST contains a list of all the files to be included. Without the @, only the OTHERS.LST file would be included.



You may include other files in this dependency list by typing them in. Be sure to follow the format shown when you enter files by editing the text file.

Source	Destination
C:\EXAMPLES\A099.BN1 ;	\PAGE.BN1

It is possible to create up to six subdirectories by editing the text file. For example, if you wanted to put the PAGE.BN1 from the above example in a subdirectory called PAGE, you would enter the following as your destination.

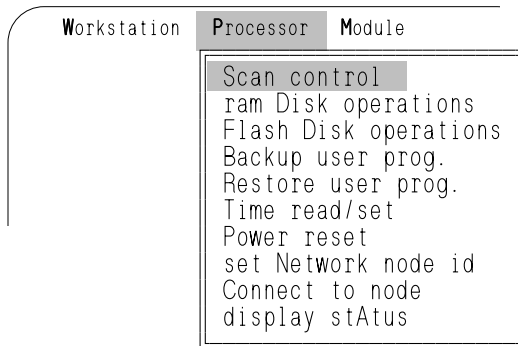
Source	Destination
C:\EXAMPLES\A099.BN1 ;	\PAGE\PAGEBN1

```
===== ex.lst =====  
# Dependency file for EXAMPLE.LDO  
# Format:  
# Source (Workstation) File Spec. ; Destination (Control) File Spec.  
#  
#  
# Program Source  
#  
C:\EXAMPLES\EXAMPLE.LDO ; \EXAMPLE.LDO  
C:\EXAMPLES\EXAMPLE.REM ; \EXAMPLE.REM  
#  
# UDFB Source  
#  
C:\EXAMPLES\PAGEUDFB.LDO ; \PAGEUDFB.LDO  
C:\EXAMPLES\PAGEUDFB.REM ; \PAGEUDFB.REM  
#  
# UDBF Libraries  
#  
C:\EXAMPLES\PAGE.LIB ; \PAGE.LIB  
#  
# List of Other files  
#  
# Contents of OTHERS.LST  
#  
# Other files required for page example LDO  
#  
# Source (Workstation) File Spec. ; Destination (Control) File Spec.  
#  
#  
#  
C:\EXAMPLES\A099.BN1 ; \PAGE.BN1  
C:\EXAMPLES\FILE1.TXT ; \FILE1.TXT  
#  
#  
C:\EXAMPLES\FILE2.TXT ; \FILE2.TXT  
=====
```

Module Menu - Find Duplicates

MI(Module - find duplicates) This command allows you to search through your LDO for coils, variables, or function blocks that are used more than once in the LDO. If a duplicate coil, variable, or function block is found, a message appears and you will be asked if you want to continue.

Processor Menu - Scan Control



P (Processor) The Processor menu contains six commands shown on the left. They will all be covered in the next six sections starting with Scan control.

When the PiC900 is running, the CPU repeatedly scans the application program in its memory. A scan is the cyclical process of:

1. Reading data from the hardware input modules
2. Using the forcing list to modify input data
3. Executing ladder logic and using the data to update program variables
4. Using the forcing list to modify output data
5. Sending commands to the hardware output modules
6. Sending data to the computer workstation (to be displayed in PiCPro animation and/or view modes)

NOTE: When you created your application program, the PiC900 hardware modules were declared under Hardware declarations. Each input/output point was assigned to a specific logic element through the Software declarations.

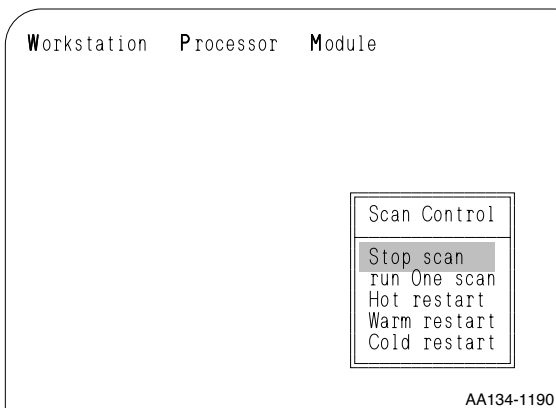
Step 3 above is called the "ladder scan", and steps 1, 2, 4, 5 and 6 are called the "system overhead." How long a scan takes depends on the length and complexity of the program.

The PiC900 system shuts down as a safety measure if either the ladder scan or the overhead takes more than 200 ms.

WARNING

Do not use any commands in the Scan Control menu until you understand how the PiC900 system runs and exactly how the command will affect the application. Starting or stopping the scan at random may:

- cause injury
- damage the machinery



PS (Processor - Scan control) gives the Scan Control menu.

PSS (Processor - Scan control - Stop) Turns the scan off. The current scan is completed, but the logic commands sent to the output modules are all 0. Power is still on to the system, so all elements in the program - timers, counters, input contacts, output coils, and so on - keep their values until the scan starts again.

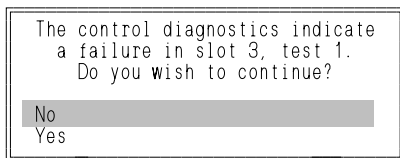
PSO (Processor - Scan control - One scan) The CPU runs through one complete scan and then sends zeros to all outputs.

PSH (Processor - Scan control - Hot restart) Restarts the scan, keeping element values as they were when the scan was stopped. The application continues as if the Stop scan command had not been sent.

PSC (Processor - Scan control - Cold restart) Restarts the scan with all elements returned the initial values Declared by the programmer, as if the module had just been downloaded.

PSW (Processor - Scan control - Warm restart) Restarts the scan. All elements that were Declared by the programmer to be "retentive" keep the values they had when the scan stopped. All other elements return to the initial values they had when the module was first downloaded. Scanning resumes and the application continues normally.

NOTE: If one of the hardware modules fails to pass its diagnostics tests when power is cycled, its DIAG LED remains lit. The system will not begin its scan cycle when you turn on the CSM Run key.



If you then type **PSC** to start the scan from the workstation, this message appears on screen, to advise you of the hardware failure. Press <Enter> to acknowledge the warning, then change the module according to the guidelines in the Hardware manual.

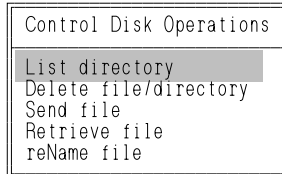
"Test 1" simply indicates which test was unsuccessful. If you have repeated failures and have to call Giddings & Lewis, the test number gives an indication as to what is wrong.

It is possible, though not advisable, to start the system from the workstation without replacing the defective module. If the PiC900 system is not connected to an application, you could type Y at the "Diagnostics failure" pop-up. The scan LED goes on, and the hardware inputs and outputs function as normally as possible, depending on the type of failure.

Processor Menu - Ram Disk Operations

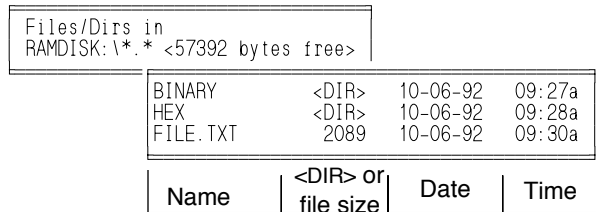
A RAMDISK is a hardware option which makes extra memory available to the PiC900 system. When working from the current disk or directory on the workstation, RAMDISK files can be listed, deleted, sent, retrieved, and renamed. They are then available to the PiC whether the workstation is connected or not.

See I/O function blocks for information on accessing the RAMDISK from the application program and the workstation files.



PD (Processor - ram Disk operations) This menu allows you to handle data files in additional memory on RAMDISK where the PiC900 CPU can access them as needed.

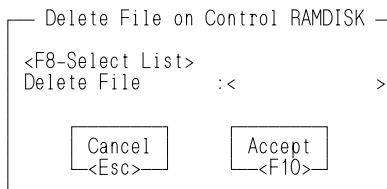
The first command lists any directory/file on the RAMDISK and the amount of disk space available.



AA1140-4292

PDL (Processor - ram Disk operations - List) A listing of the directories/files stored in the RAMDISK will appear. In this example, the RAMDISK contains two directories and one file. There are 57392 bytes of disk space available on the RAMDISK.

To bring up a list of the files within a directory, highlight the directory and press <Enter>.



AA1139-4292

PDD (Processor - ram Disk operations - Delete)

Pressing <F8> will bring up a list of directories/files on the RAMDISK. Select the file you want to delete from the list and press <Enter>. Press <F10> to delete the file or <Esc> to cancel.

NOTE: You must delete all the files within a directory before you can delete the directory. You must type the name of the empty directory in the Delete File field in order to delete it. It cannot be selected from the <F8> list. When you select a directory from the list, it displays a list of the files within the directory.

PDS (Processor - ram Disk operations - Send) Any file/directory can be copied into the RAMDISK from the workstation under the same or a different name.

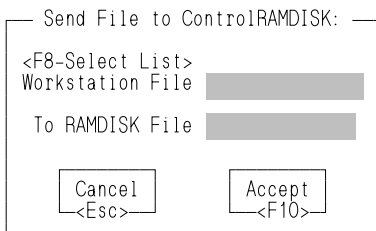
Enter the name of the directory/file either by typing the directory/filename/extension or by pressing <F8> and choosing the file you want from the selection box.

Move the cursor to the second entry field and enter the destination for the file (directory/filename/ extension) in the RAMDISK using the same name or a different one. Pressing <F8> lists the directories/files already in the RAMDISK.

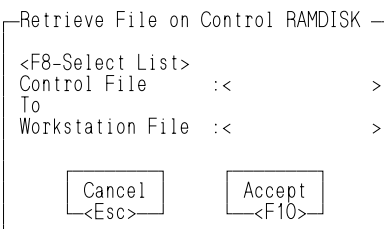
Press <F10> to send the file or <Esc> to cancel.

PDR (Processor - ram Disk operations - Retrieve) A file can be retrieved from the RAMDISK and sent to the workstation. Again you can keep the RAMDISK name or change it.

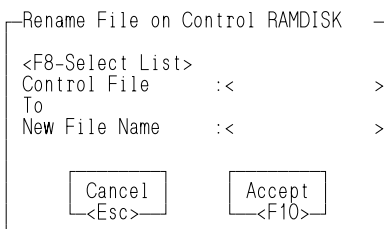
Press <F10> to retrieve the file or <Esc> to cancel.



AA162-1790



AA1137-4292



AA1138-4292

PDN (Processor - ram Disk operations - reName)

Allows you to rename a file/directory on the RAMDISK.

Press <F10> to rename the file or <Esc> to cancel.

Processor Menu - Flash Disk Operations

When the FMSDISK hardware option is installed on the PiC CPU, files can be stored on the FMSDISK. These files can then be accessed from the LDO using some of the functions in the COMM group.

FMSDISK files can be listed, retrieved, and updated using the commands found under the Processor menu.

```
Control FlashDisk Operations
List directory
Retrieve file
Update Disk
```

PF (Processor - Flash Disk operations) This menu allows you to handle data files on the FLASHdisk.

The first command lists any directory/file on the FMSDISK and the amount of disk space available.

```
Files/Dirs in
FMSDISK:\*.* <57392 bytes free> |
PAGE          <DIR>  10-06-92  09:27a
FILE.LDO      2089   10-06-92  09:30a
```

Name	<DIR> OR file size	Date	Time
PAGE	<DIR>	10-06-92	09:27a
FILE.LDO	2089	10-06-92	09:30a

PFL (Processor - Flash Disk operations - List directory) The file or files on the FMSDISK can be listed using this command.

```
Retrieve File from Control
<F8-Select List>
Control File   :<      >
To
Workstation File :<      >
```

Cancel <Esc> Accept <F10>

PFR (Processor - Flash Disk operations - Retrieve) A file can be retrieved from the FMSDISK and sent to the workstation.

Press <F10> to retrieve the file or <Esc> to cancel.

Processor Menu - Backup User Prog.

```
Update FLASH Disk
<F8-Select List>
File Name      :<      >
To Flash Disk File:<      >

Cancel        Accept
<Esc>        <F10>
```

PFU (Processor - Flash Disk operations - Update)

Enter the filename if sending an individual file. If the file you enter here is a list of several files, use the @ in front of the filename to include all the connected files in the dependency list.

The @ indicates that FILENAME.LST contains a list of all the files to be included. Without the @, only the FILENAME.LST file would be included.

Press <F10> to update the FMSDISK with a file or a list of files or <Esc> to cancel.

PB (Processor - Backup) A program in PiC900 memory may be Backed up as a binary file directly to the computer disk. When you type its filename in the dialog box, you must include the extension .BIN so you can use the Restore command later.

NOTE: If you are working with an LDO on your workstation, the filename of that LDO with a .BIN extension appears in the entry field.

Processor Menu - Restore User Prog.

PR (Processor - Restore) A computer file in binary format with extension .BIN may be Restored to PiC900 memory. The Restore option may be used whether you used Backup or Bin-86 dump command to create the file. When you type the name in the dialog box, you must type its extension as well.

Processor Menu - Time Read/Set

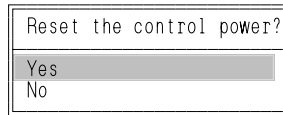
```
Adjust Control's time & date
Enter time (HH:MM): 08:10
Enter date (YY-MM-DD): 92-06-10
<F8> for DOS time and date

Cancel        Accept
<Esc>        <F10>
```

AA136-1190

PT (Processor - Time) The PiC900 system has its own battery-powered clock located in the CSM.

If the date or the time in this dialog box is wrong, highlight it and press <F8>. The correct value is filled in from computer memory. When the dialog box is accepted, the PiC900 clock is set with its values.



AA137-1190

PP (Processor - Power reset) This command has the same effect as using the switch on the CSM to cycle power. A pop-up asks you to confirm the command.

When power is cycled, either with this command or with the CSM switch, the scan resumes with a warm restart (see Warm restart above under Scan control).

WARNING

WARNING

Do not reset PiC power until you understand how the system runs and exactly how power loss will affect the application. Starting or stopping an application at random, may:

- cause injury
- damage the machinery

NOTE: When you cycle power, communication between the computer and the PiC is cut off. If you are using *Animate* or *View Variables*, cycling power turns the option off. A Communication Error message appears. Press any key to erase the message, then toggle the option off and on again.

Also make sure any files you have open on the RAMDISK or the workstation if connected are closed.

Peer-to-Peer/PiC(Servo)Pro Communications

Peer-to-peer communication using twisted pair wire can be set up between PiCs whose CPUs have network hardware. See the hardware manual for details.

If you want to communicate to networked PiCs through PiC(Servo)Pro in order to program, monitor, force, or tune, you will use the next two commands found under the Processor menu.

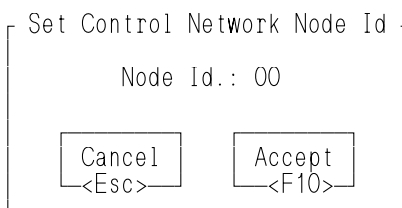
Processor Menu - Set Network Node ID

Each PiC on the network must be given a unique node ID number. The range of numbers available is 1 to 255 excluding 65 (41 Hex). 65 is reserved to identify the PiC physically connected to the PC through the RS232 programming port. This is the local node. All other PiCs on the network are called network nodes.

Setting the node to zero clears the ID and removes the PiC from the network.

NOTE: The system node number set with this command has priority over any node number set from the LDO using the NETOPN function block. If you are using the Network function blocks, the source ID number entered in the NETOPN function block must match the ID number assigned with this command. If it does not, an error is indicated. A transitional contact must be used to enable the Network function blocks.

If no system network node id number is set, then any ID can be set by the NETOPN function block.

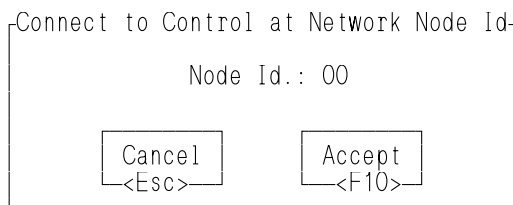


```
Set Control Network Node Id
Node Id.: 00
Cancel <Esc>   Accept <F10>
```

PN (Processor - set Network node id) This command allows you to set the network node ID for this node.

Initially, you must connect the PC to each PiC through the RS232 programming port to set the network node ID. The node ID will not have to be set again unless you want to change it or unless the CPU module in the PiC is changed. The scan is stopped when you set the node.

Processor Menu - Connect to Node



```
Connect to Control at Network Node Id
Node Id.: 00
Cancel <Esc>   Accept <F10>
```

PC (Processor - Connect to node) Once a node ID has been assigned to every PiC on the network, this command allows you to connect the PC to any PiC on the network.

```
Control EPROM Version  
Program Downloaded  
  
Scan is Stopped  
Run/Stop switch in Run Position  
Forcing is Disabled  
No Run Time Failures  
No Hardware Failures.  
CPU Type      :  
RAMdisk       :  
FMSdisk       :  
Network Node ID :  
  
Press any key to continue
```

PA (Processor - stAtus) This command allows you to display the status of the control you are connected to.

The display status box provides the following information.

- Line 1 Lists the version number of the EPROM installed in the PiC CPU.
- Line 2 Lists the name of the LDO currently loaded into the control and the date and time the LDO was last downloaded.
- Line 3 Tells you whether the scan is running or stopped.
- Line 4 Tells you the position of the Run/Stop switch.
- Line 5 Tells you whether forcing is enabled or disabled.
- Line 6 Lists run time failures that also appear on the CPU diagnostic LED.
- Line 7 Lists hardware failures of modules in the rack(s) whose diagnostics have failed.
- Line 8 Lists the type of CPU in the control and the amount of application memory.
- Line 9 Lists the size of RAMdisk memory if installed.
- Line 10 Lists the size of FMS disk memory if installed.
- Line 11 Lists the network node ID of the PiC if it has been set.

Troubleshooting an Application

If a program does not work properly, there may be a bug in the software module that is running the PiC900 system. However you cannot be sure of that until you eliminate the system hardware, the I/O connections, and the application as possible causes of the difficulty. If you reach the last step in Table 4-2, you may assume the error is in the software module.

Checklist in Troubleshooting

Checklist	What should happen	If it doesn't
Check the "fuse blown" (FB) LEDs on output modules and SCAN LED on CSM module.	FB LEDs on output modules all off. SCAN LED on CSM is lit.	Turn off RUN/STOP key and main disconnect. Replace the fuse. Turn on power. Turn on the RUN/STOP key to restart the scan.
Cycle power on the system to see if the modules pass their diagnostics tests.	DIAG LEDs all go out.	Turn off power at the main disconnect. Unscrew the terminal connector on the faulty hardware module, replace it with another of the same type, and screw the connector on the new module. Turn on power.
Check the Software Declaration table to see which element names correspond to the problem outputs. Use the Element-Find command to bring a coil up on screen, and turn on Animate. If the logic value in the coil corresponds to the voltage at the screw terminal, Force the opposite value into the coil. Repeat with other possible problem outputs.	The outputs respond properly to logic commands.	Turn off power at the main disconnect. Follow the procedure above to replace the output module.
Find and Animate the contacts that correspond to input points. At each input screw terminal turn the voltage on and off while watching the logic state of the corresponding contact.	The contacts have the correct logic states.	Turn off power at the main disconnect switch. Follow the procedure above to replace the input module.
Turn input devices on and off and see how the input elements respond. Also Force 1s and 0s into output elements and watch how the corresponding devices respond.	Continuity and application devices check out.	Correct problems.
Force the input elements one at a time and see if the output elements react as expected.	The outputs work correctly.	Use Animate with the Find commands to trace the logic flow. Force variables at various points in this flow to isolate networks that should be edited.

CHAPTER 5 Motion Control Concepts

This chapter presents an overview of motion control with the PiC. The focus is on closed loop servo control where the PiC receives/sends information from/to a servo axis.

The PiC can also be used with a “read only” or digitizing axis where it receives position information from the axis but does not send information to the axis. Often, this axis is used as a master to another servo axis.

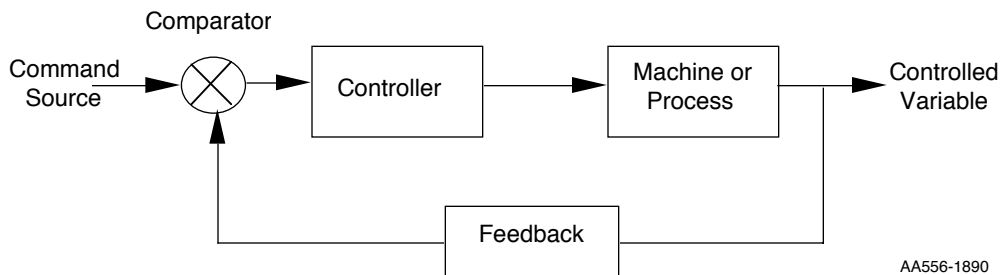
Background Information on Closed Loop Control

Closed loop control is characterized by the ability to compare the actual value of a controlled variable (i.e. axis position) to its desired or commanded value and automatically take action based on that comparison. It is a self-correcting system.

In Figure 5-1, the command source provides a desired value. The actual output value of the machine or process is sent back and compared to this commanded value. The difference between the commanded and actual value represents an error. The controller computes command signals to send to the machine or process to correct for the error. The machine or process acts as the final correcting device to bring the actual output as close as possible to the commanded input while maintaining a stable system.

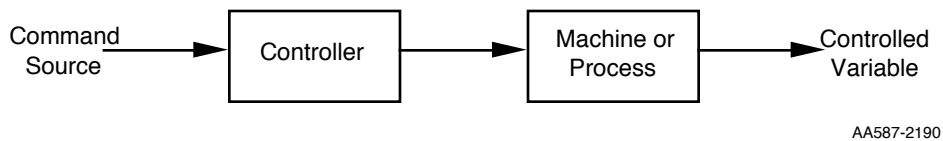
When the controlled variable being monitored and corrected is the position of an axis, the system is referred to as a *servo* system.

FIGURE 5 - 1. Closed Loop System



An open loop control system (Figure 5-2) is not self-correcting. A command is issued to the controller and action is taken. The controlled variable output is controlled by system inputs only. Because there is no feedback, no account is taken of the actual system output.

FIGURE 5 - 2. Open Loop System



The majority of information presented in this chapter focuses on closed servo loop control.

Introduction to Motion Control with the PiC

Figure 5-3 gives an overview of a motion control system with the PiC. A summary of the components follows.

1. The system is controlled by the **PiC**.
GLOS (Giddings & Lewis operating system) is a real time multitasking operating system on the CPU designed for maximum efficiency in motion control applications. The PiC provides motion control for up to 16 axes.
2. When the PiC is used for motion control, PiCServoPro software on a **personal computer** is required. PiCServoPro includes three program development tools:
 - PiC programmer (PiCPRO)--the off-line system used to develop ladder programs for an application
 - Servo setup and tuning (SRVSETUP)--the Servo setup program used to set up axes and tune servo loops
 - Axis profile setup (PiCPFL)--the PiC Profile program used to build ratio profiles for master/slave applications

The personal computer provides the platform to:

- use PiCServoPro software to create, edit, and print application programs
- perform high-speed serial communication with the PiC for downloading the ladder program, tuning the servo loop, running scan control, monitoring the ladder program, viewing variables, and forcing variables

Refer to Chapter 1 of this manual for the procedure to install PiCServoPro on your IBM or compatible PC.

3. Refer to Chapters 2 and 3 of this manual for information on programming with PiCPro.
4. **Machine control inputs and outputs** required for the application

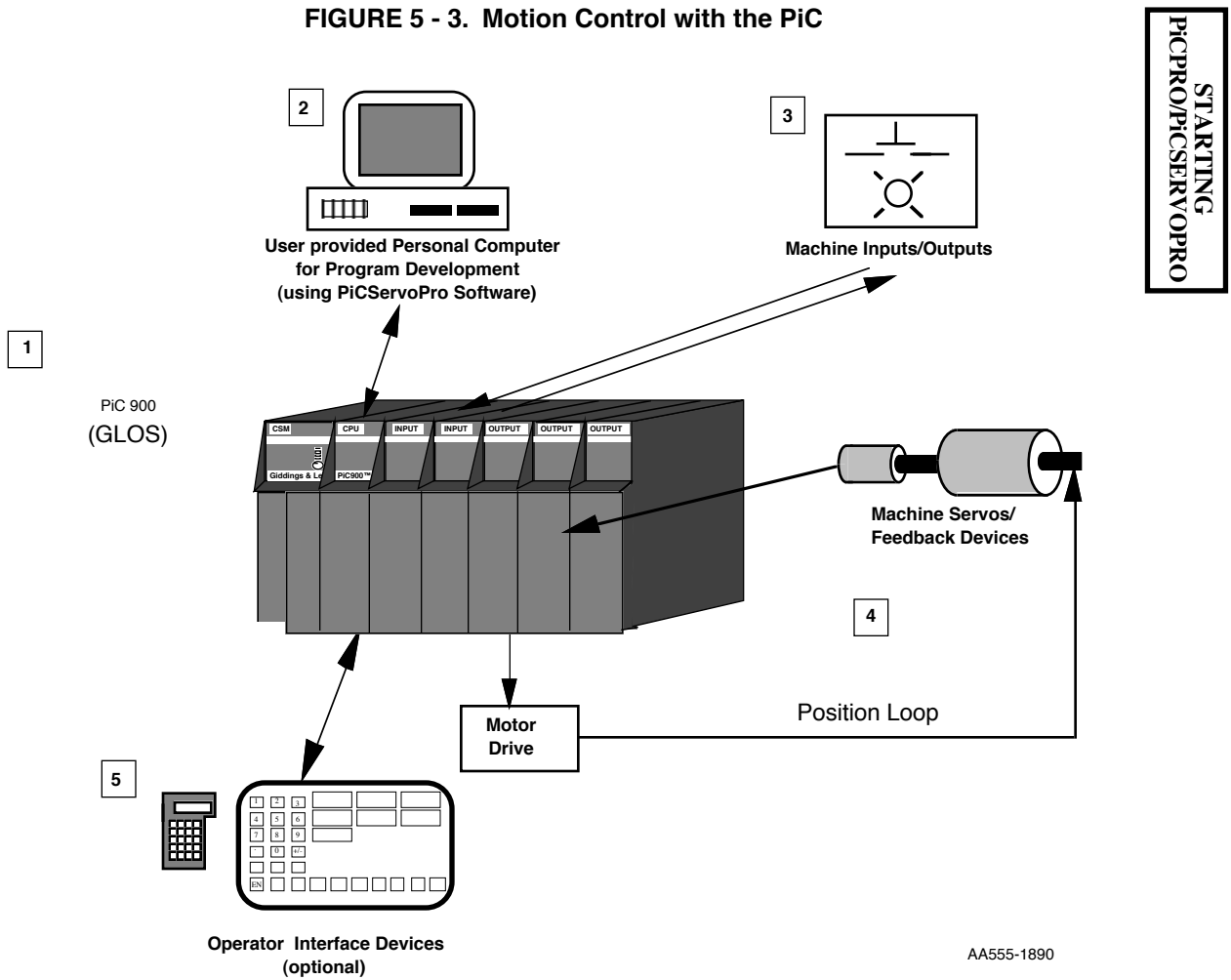
The **position loop** which includes:

- The feedback device on the machine sending position information to the PiC via the feedback module

NOTE: If the system has a master rack and one or more expansion racks, feedback modules must always be installed in the master rack; not in an expansion rack.

- The PiC sending signals to the motor drive via the analog output module
5. The optional **operator interface devices** (Fluke touch screen, PiC Programming Pendant)

FIGURE 5 - 3. Motion Control with the PiC



In order to control a position loop, the PiC must:

1. Read feedback information on the actual position of the axis from the feedback module.
2. Perform calculations on servo data.
3. Write command signals to the analog output module to correct for any error in position.

Calculations are performed on the servo data by interrupting the ladder scan, performing calculations, issuing commands, and then continuing the ladder scan. By doing this on an interrupt basis, the servo calculations can be done on a constant time interval and more frequently than if they had to be done in the ladder. This provides for much finer resolution in position control.

Figure 5-4 gives a detailed picture of how the servo software in the PiC handles servo data with a position loop.

The command source comes from the ladder program you create. Specifically, commands such as feedrate, distance, endpoint, etc. are contained in the standard move functions incorporated in your ladder. The servo software accesses this data from a queue.

There are two queues available:

- The *active* queue which holds the move that is currently being executed for the specified axis.
- The *next* queue which holds the move that will be executed when the active move is completed or aborted for the specified axis.

Using queues allows the moves to be executed faster than if they were dependent on the running of the ladder. It also keeps the time between the first move and the second move for the specified axis as short as possible.

The queues are transparent to you for the most part. They can, however, be manipulated with standard functions as described in the Function/Function Block manual. An identification or “que” number is assigned by the software to each move when the OK output on the move function is set. The sequential range of que numbers is from 1 to 255.

During iteration, the servo software makes the necessary calculations and generates command segments. These command segments are fed into the position loop (1) on a real time interrupt rate designated by you. A typical choice is 4 ms. Every 4 ms or interrupt the command segment is added to the command position. Typically, the servo loop is executed eight times for each update received from the iterator. When running in master/slave applications, iteration occurs every interrupt.

The actual position being read from the feedback module is compared to the command position. A following error is calculated from the difference between the command position and the actual position.

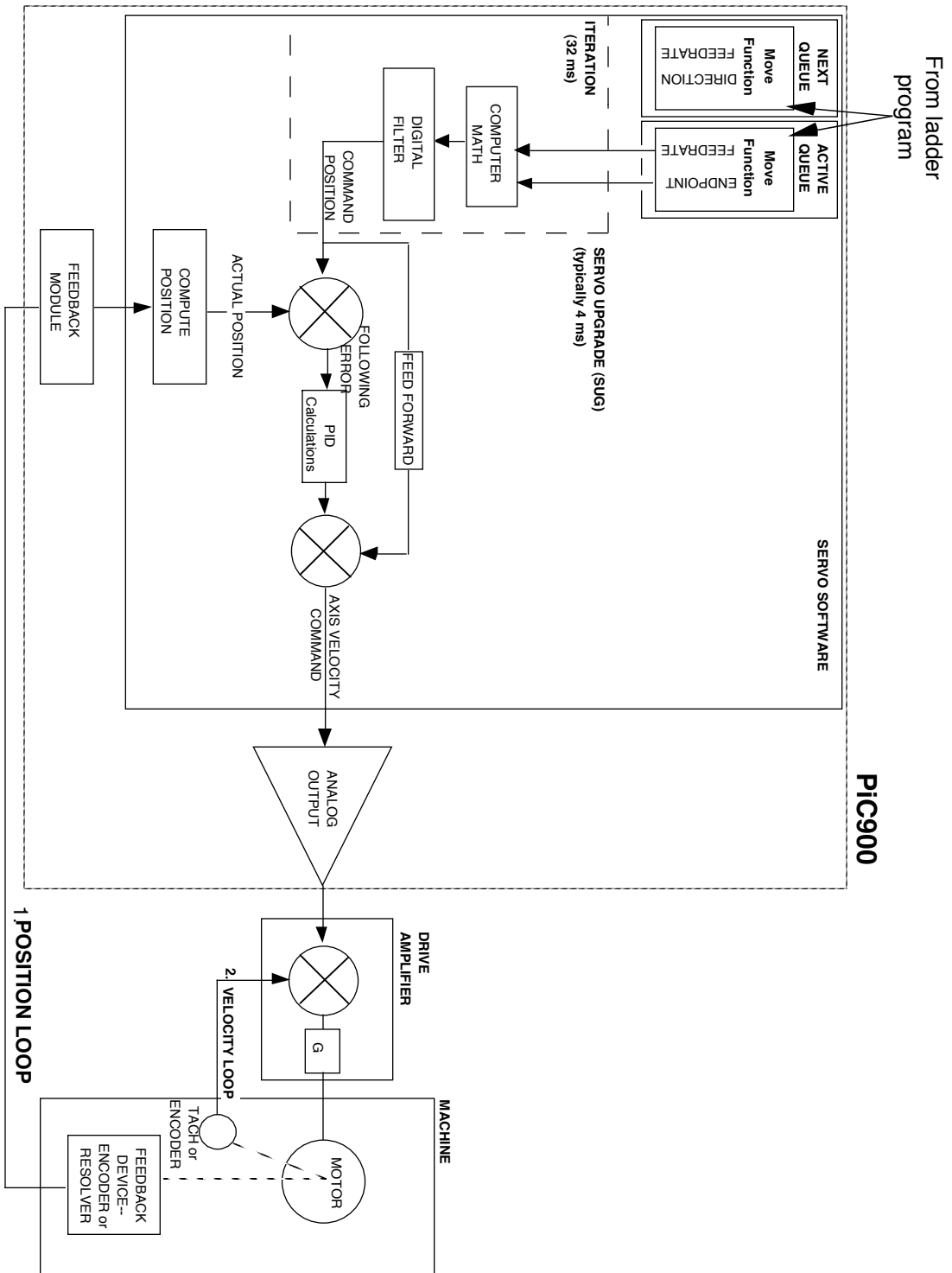
This error is then fed into the PID (proportional, integral, derivative) loop which converts it to a proportional digital velocity command. The PID loop is a compensation network used to improve the output of the position loop. Any or all of the components of the compensation network can be used in an application:

- The *proportional* control causes the output to change in proportion to the value of the following error.
- The *integral* control causes the output to change in proportion to the value of the error over a period of time at a zero velocity command.
- The *derivative* control causes the output to change in proportion to the rate of change of the error.

Feed forward is also part of the compensation network. It corrects for some or all of the following error.

The Analog Output module converts the digital velocity to an analog velocity command and sends it to the drive/motor unit.

FIGURE 5 - 4. PiC Position Loop Control



AA579-2090

NOTE: There is also a velocity loop (2) in the control system independent of the PiC. The velocity loop ensures a constant motor speed and a tight servo loop. An encoder or tachometer provide the feedback signal. When an outside load disturbance causes deviations in the velocity, the loop allows for instantaneous compensation. It controls velocity only, not position. It is internal to the drive and motor package.

Basic Motion Control Programming Process

To accomplish motion control for your application, there are some general considerations presented here. They are designed to give you an overview of what needs to be done. Your individual application will dictate the actual logic needed in your program.

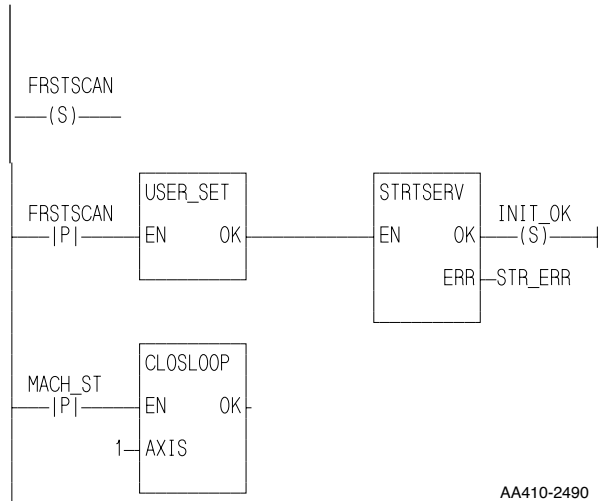
When developing your motion control program, there are three general areas to consider.

1. Preparing for axis motion

The standard motion function **STRTSERV** [along with the user-defined setup function (**USER_SET**) which will be explained later] is required to initialize setup data, run the servo software, and start interrupts.

The loop is closed with the **CLOSLLOOP** function. (See setup section that follows.)

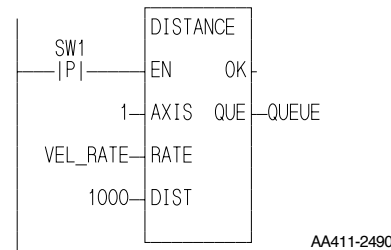
The servo software is now ready to accept commands for motion.



2. Performing axis motion

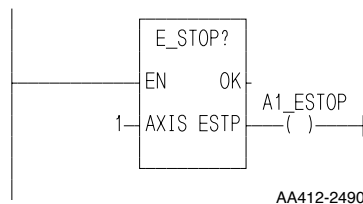
By entering any move function, the servo software is told to perform certain tasks. These can be as simple as jogging an axis or as complex as performing a ratio profile in a master/slave application.

NOTE: Use a transitional contact to start each move type.



3. Evaluating axis motion

This involves programming that will check for errors, read servo data, and make appropriate decisions based on the information it receives.



Setup data

Before the loop can be closed for any axis, there is axis setup information the PiC needs in order to know how your system is setup. The USER_SET function shown in (1) above is made by you with the Servo setup program and holds all the setup information for all the axes used in your application. The procedure for doing this is covered in Chapter 6 of this manual.

The setup data is divided into three sections. These sections and their related data are listed below.

1. **Scaling data** - tells the servo software how the axis units (inches, millimeters, degrees, etc.) and ladder units (integers used in the ladder program) relate to the feedback units the software uses to perform its calculations and issue its commands.

Input scaling - Feedback units, ladder units, and axis units for data manipulation

Output scaling - Commanded voltage, motor RPM, and counts/motor revolution to scale value for the analog output module

2. **Iterator data** - provides the servo software with data on how the axis should be handled by the move iterator.
3. **Position loop data** - provides the servo software with data it needs to control the position loop for the axis.

Velocity limit

Acceleration ramp

Deceleration ramp

Controlled stop ramp

Slow velocity filter

Fast velocity filter

Slow/fast velocity threshold

Rollover on position

Rollover position

Software upper limit

Software lower limit

Ignore limits until referenced?

Input polarity

Output polarity

Analog output offset

Feed forward percent

Proportional gain

Integral gain

Plus integral error limit

Minus integral error limit

Derivative gain

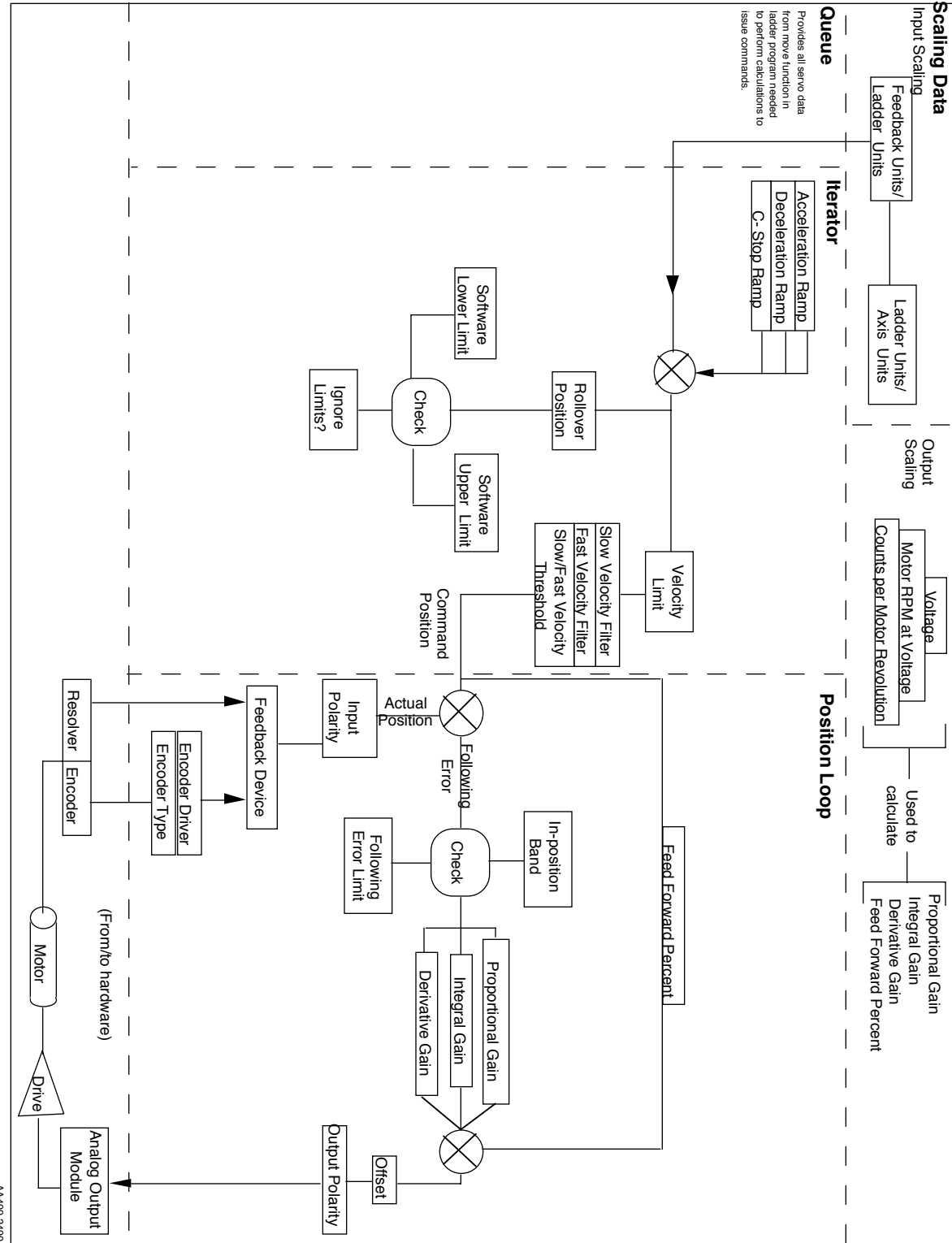
Following error limit

Update rate

In position band

Figure 5-5 illustrates how the setup data is used by the servo software to control motion.

FIGURE 5 - 5. Setup Data used by Servo Software



AA403-2490

CHAPTER 6 Servo Setup and Tuning

Servo setup and tuning is a program development tool used to:

1. Enter setup data for all axes used in your application with the servo setup portion.
2. With communications established between the PC and the PiC, change six of the setup parameters and view on-line variables and errors for the selected axis with tuning (PiCTune).

This chapter covers both of these features of the Servo setup program.

Introduction to Servo Setup Data

For each axis used in your application, certain setup parameters must be entered and included in your program. This is done with the Servo setup and tuning (SRVSETUP) program found under the Programs menu.

There are three categories of setup data in the Servo setup program:

1. **Scaling data** - sets up the ratio between feedback units, ladder units, and axis units. This allows you to enter axis units instead of feedback units in the appropriate places for iterator and position loop data in setup. The three types of units are defined below.

Feedback units - the units the servo software uses to perform its calculations and issue its commands in.

Ladder units - the units used in the ladder program. They must be integers.

Axis units - the units of measurement (inches, millimeters, degrees) for the system. They may be integers or non-integers (decimals) and are used in the Servo setup program to enter certain types of setup data.

2. **Iterator data** - specifies how data such as limits, ramps, filters, and roll overs will be handled by the move iterator.
3. **Position loop data** - provides the servo software with information on how the axis is set up for the position loop.

Figure 6-1 illustrates how the setup data interacts with PiCPro on the PC and the CPU on the PiC. A general overview of using the Servo setup program with PiCPro and the PiC follows.

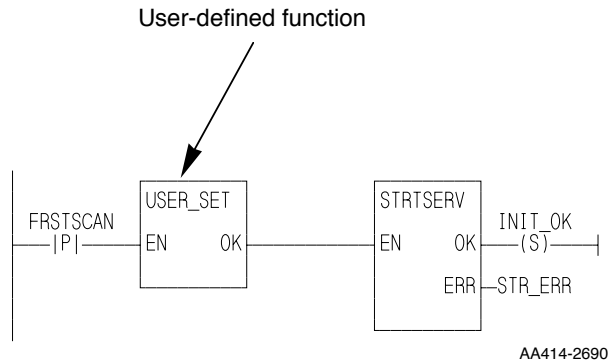
1. Use the Servo setup program (SRVSETUP.EXE) to enter setup data for your application. Save this data in a file (*your name*.SRV). [USER.SRV in Figure 6-1]
2. Use the Servo setup program to make a function containing this setup data. Assign an appropriate name. (USER_SET is the name of this function in the example ladder network at x(6) in Figure 6-1.)

This function is placed in a library file you create (*your name*.lib) which PiCPro can find. The library file holds servo functions defined by you (such as

the USER_SET function in Figure 6-1) for your application. [ZSERVO.LIB in Figure 6-1]

The procedure to do this will be explained in section 6.3 of this chapter.

3. Use the PiCPro software (PICPRO.EXE) to create an application program (*your name*. LDO). [USER.LDO in the example]
4. Include the setup function you made in a network of your ladder program.

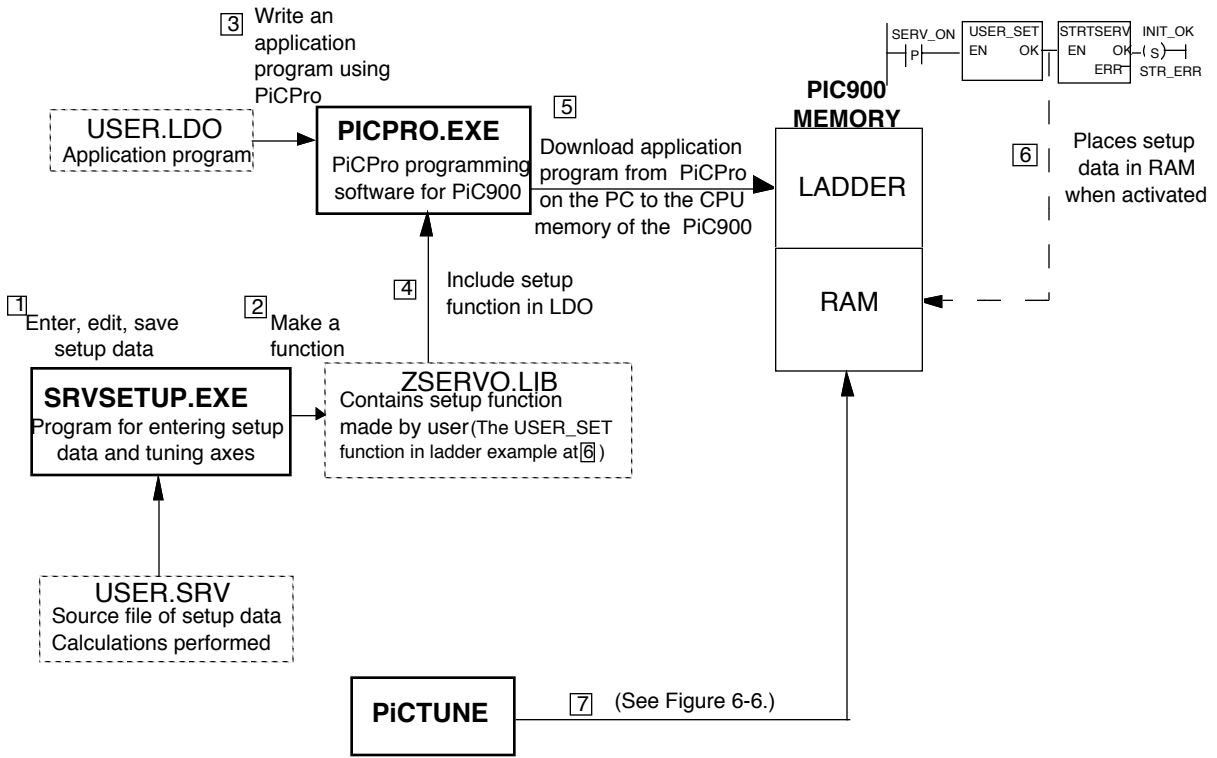
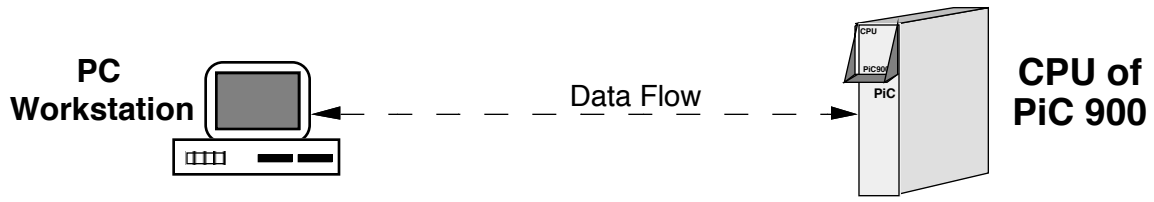


5. Download the application program to the PiC. The setup data for your application is sent to the PiC in the form of a function.
6. When the USER_SET function, in combination with the STARTSERV standard motion function, is called in the ladder, the setup data is copied into the RAM memory of the PiC.
7. PiCTune is a part of the Servo setup program used to read and write certain setup data for each axis. It will be covered in section 6.4 of this chapter.

WARNING WHEN UPGRADING PICPRO

Always remake your setup function and download your LDO when you upgrade your PiCServoPro software to the latest version.

FIGURE 6 - 1. The Servo Setup Program Data Flow



Key

Represents data files created by user

Represents executable files supplied with PiCServoPro

AA576-4992

Entering Servo Setup Data

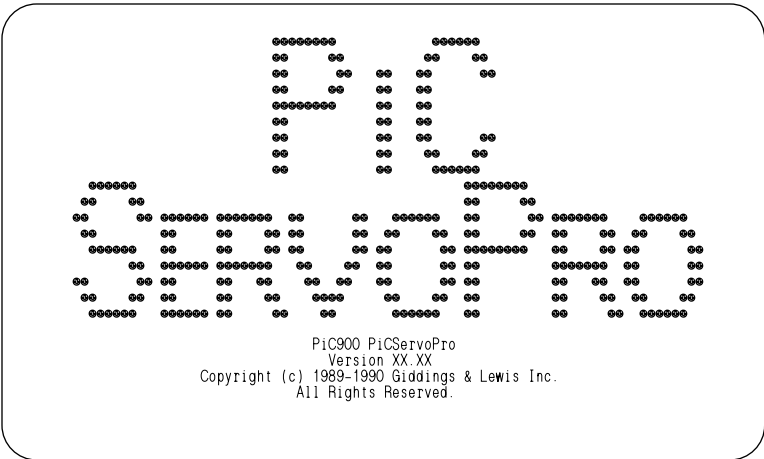
The procedure for entering setup data for each axis used in your application is covered in this section.

This is done from the Servo setup and tuning (SRVSETUP) menu under Programs.

NOTE

The same methods of moving around within PiCPro discussed in an earlier chapter apply to PiCServoPro also. (Hot keys, menus, etc.)

At any point, you may use the <F9> key to receive additional information on the item the selection bar is on.

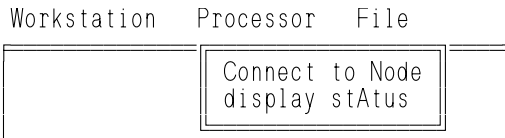
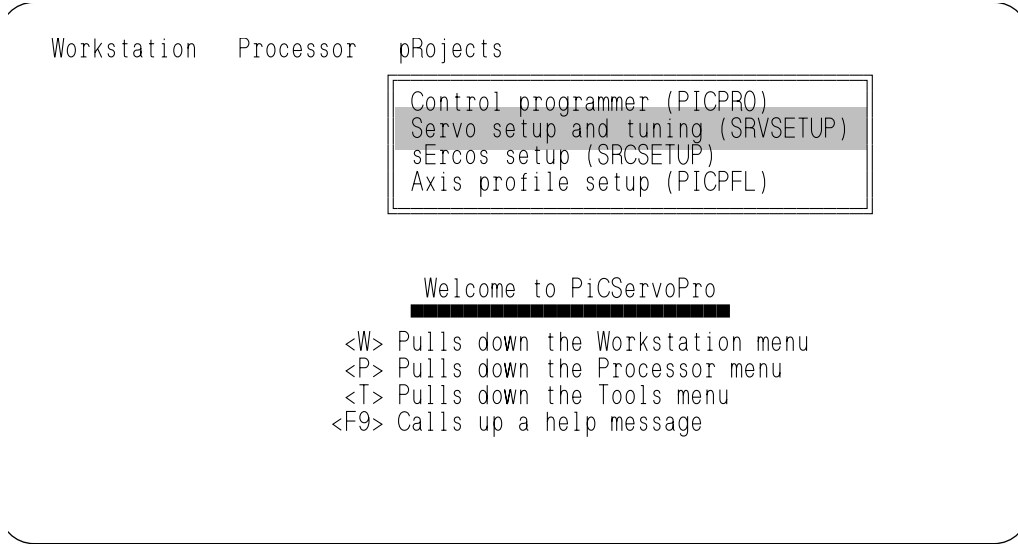


AA506-1490

When PiCServoPro is called up on the PC, the screen shown on the left appears momentarily (hit any key to move to next screen) before the menu bar containing Workstation and Programs appears.

The Workstation menu offers the same items as explained in Chapter 1. In addition there is a Help on PiCServoPro message.

Accessing Programs brings up the three choices shown below. To enter setup data, place the cursor on Servo setup and tuning (SRVSETUP) and press <Enter>.



The menu bar on the left will appear. The Processor menu allows you to use the servo setup and tuning program over your network.

Press **P** (Processor) to display the choices under Processor. If you want to connect to another node on your network, press **C** (Connect to Node).

If you want to display the status of the control you are connected to, press **A** (display of stAtus).

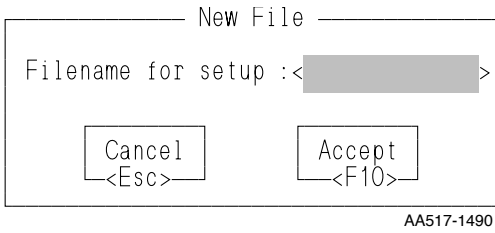
NOTE: In the right bottom of the screen, the number of the connected node appears.



Press **F** (File) to display the choices under File. If a new file for setup data is being created, press **N** (New).

If a file that was created previously is being worked on, press **O** (Open).

NOTE: When opening existing files, the F8 key displays a list of any files that have been previously created.



When creating a new file, this box appears and you must type in a name for your setup data file. Press F10 to accept the name and continue.

NOTE: After F10 is pressed, notice that the name of your setup data file with the extension .SRV appears on the bottom of the screen. This file will hold all the setup data for all axes used in your application.

The menu bar will change as shown below and you will be prompted to add an axis by choosing “insert” from the Axis menu.

Workstation Processor File Axis

```
.....To add an axis, select `Insert' command from the `Axis' menu
<End of axis table>
```

Press **AI** (Axis - Insert) or the <Insert> key.

```
Workstation Processor File Axis
= Label == Axis # ==
|| Insert axis <Ins> ||
|| Redefine axis <Ctl-R> ||
|| Delete axis <Del> ||
|| Feedback data <Ctl-F> ||
|| Edit axis data <Enter> ||
|| Tune axis <Ctl-T> ||
```

The Define Axis box appears. The Feedback Module field has been added.
NOTE: The same box appears if you choose to redefine an axis after it has been inserted.

AA518-3692

In the **Define Axis** box, the Axis label is entered automatically. You can change this label. You must provide the hardware information based on your application.

The types of axes that can be defined include:

Servo application

A servo axis controlled by the PiC. Fill in all information.

Digitizing axes

An input-only axis (Fill in only Feedback Module and Input slot and channel. Axis #'s 49 through 64 are reserved for this use.)

The Feedback Module field allows you to enter the feedback device used for the axis. Press <F8> to bring up a list of types. Move the cursor to the feedback device you are using and press <Enter>.

Fill in the appropriate slot/channel information and press <F10>.

NOTE: When SERCOS is the feedback type, enter the ring number in the channel entry.

NOTE: When Stepper is the Feedback type, the input slot/channel entry must match the output slot/channel entry.

Press <Esc> to cancel data

Depending on the feedback module chosen, one of the boxes shown below will appear after you press <F10> so that you can enter setup data for the module. You can also change any data entered in these Configure boxes from the Axis menu using the Feedback data command <CTL - F>.

Encoder Setup

Each box has a default setting in it. Pressing <F8> will bring up a list of available choices for some of the entries. With the cursor on an entry, press <F9> to get Help messages.

```
Configure Encoder Module
Encoder Driver :<Differential >
Encoder Type  :<Quadrature  >
Cancel
<Esc>
Accept
<F10>
```

AA1120-3692

Enter the correct driver for the encoder being used.

Single-ended or Differential

Enter the encoder type being used in your application. It may be either quadrature or pulse.

Quadrature type assumes that there are four counts for each quadrature cycle. The rising and falling edges of both channel A and channel B are counted.

Pulse type counts either channel A or channel B so that one count is recorded per cycle.

Resolver Setup

The resolver does not require any other setup data. An inductosyn device may also be used. Enter “Resolver” if using an inductosyn.

Analog Input Setup

```
Configure Analog Input Module

Filter Time :<1 ms      >
4/20 mA offset :<No  >
      Bipolar :<Yes >
Range (Volts) :<-10 to 10 >

Cancel      Accept
<Esc>      <F10>

AA1121-3692
```

Select the filter time from the <F8> list.

```
Analog Input Filter Time
1 ms
10 ms
100 ms
110 ms
```

If you are using the module in the 4/20 mA mode, press Y(es) at the 4/20 mA offset field. The Bipolar field will be changed to “No” and unipolar input range of 0 to 5 is inserted by the software. These defaults cannot be changed when in the 4/20mA mode.

When not using the 4/20 mA mode, the bipolar or unipolar voltage range is selected from the <F8> lists shown below.

Unipolar Ranges

```
Analog Input Range
0 to 10
0 to 5
0 to 2.5
0 to 1.25
0 to 1
0 to 0.5
0 to 0.25
0 to .125
```

Bipolar Ranges

```
Analog Input Range
-10 to 10
-5 to 5
-2.5 to 2.5
-1.25 to 1.25
-1 to 1
-0.5 to 0.5
-0.25 to 0.25
-0.125 to .125
```

TTL Setup

Configure TTL Input Module

Data Valid :<High on 24th>

of data bits :<23 >

Input unsigned/signed :<Unsigned >

Cancel <Esc>

Accept <F10>

Selections are based on the type of feedback device you are using.

Data Valid if

High on 24th IP

Low on 24th IP

Same - No 24th IP

Gray code

TTL input data is defined as valid depending on what is selected from the list.

Binary

- High** - Whenever input 24 is high, the inputs are not allowed to change.
- Low** - Whenever input 24 is low, the inputs are not allowed to change.
- Same** - Whenever two consecutive reads of the inputs are the same. The 24th input is not used.

Gray code - Alternative to binary encoding. Only one bit of a gray code number changes at a time. The 24th input is not used.

Enter the number of position data bits your device requires. A minimum of eight bits is required.

A maximum of 16 and a minimum of 8 bits can be used if Same or Gray code is selected above. The 24th input is not used.

A maximum of 23 bits can be used if High or Low is selected with the 24th bit used as an indicator of valid data.

The input unsigned/signed defaults to unsigned. Most TTL devices return an unsigned pattern with one end of travel all 1's and the other end all 0's. If the device returns a signed pattern with all 0's in the center of travel, change the default to signed by pressing <F8> and choosing Signed.

SERCOS Setup

```

Configure SERCOS Module

Slave Number 1-8 :<      >

[Cancel]  [Accept]
<Esc>    <F10>
    
```

When a SERCOS slave is connected to a servo axis, SERCOS is chosen as the feedback module. Enter the slave number in the Configure SERCOS Module box.

NOTE: Only a servo loop axis may be connected to a SERCOS slave.

After defining an axis, the menu bar displays the information as shown below. The cursor is positioned after the last entry so that pressing <Insert> allows you to define the next axis.

Label	Axis #	Feedback Module	Input Slot/Chan	Output Slot/Chan
AXIS_1	1	Encoder	3 / 1	4 / 1
AXIS_2	2	Resolver	5 / 1	6 / 1
AXIS_3	3	Analog Input	7 / 1	4 / 2
AXIS_4	4	TTL Input	8 / 1	9 / 2

There are five columns containing the information that defines the axis.

Column	Name	Description
1	Label	Represents an eight character label assigned by the servo software. It can be changed by you to an application-specific label to identify the axis throughout the Servo setup program. It only applies to the axis in setup.
2	Axis #	Represents the number assigned by the servo software to identify the axis in the ladder program. It cannot be changed by you. You enter this number at the AXIS input of any standard motion function for that axis used in your ladder program.
3	Feedback Module	Identifies the feedback device being used on the axis.
4	Input Slot/Channel	Defines the hardware input of the axis.
5	Output Slot/Channel	Defines the hardware output of the axis.

Stepper Axis Setup

Configure Stepper Module

Stepper Type: <CW/CCW>

Cancel <Esc>

Accept <F10>

Depending on the type of stepper you are using, select from the <F8> list.

Stepper Type

CW/CCW

Step/Direction

The stepper type defaults to CW/CCW.

Number of Servo and Digitizing Axes Available

It is possible to define up to 32 servo axes with input and output and up to 32 digitizing axes with input only. Figure 6-2 shows 16 servo axes and 16 digitizing axes. The servo software numbers the servo axes 1 through 16 for the first 16 servo axes and 101 through 116 for the second 16 servo axes. The digitizing axes are numbered from 49 through 80 in the **Axis #** column.

FIGURE 6 - 2. Maximum Number of Axes

Workstation Label	Processor Axis #	File Feedback Module	Axis Input Slot/Channel	Output Slot/Channel
AXIS_1	1	Encoder	3 / 1	4 / 1
AXIS_2	2	Encoder	3 / 2	4 / 2
AXIS_3	3	Encoder	3 / 3	4 / 3
AXIS_4	4	Encoder	3 / 4	4 / 4
AXIS_5	5	Resolver	5 / 1	4 / 5
AXIS_6	6	Resolver	5 / 2	4 / 6
AXIS_7	7	Resolver	5 / 3	4 / 7
AXIS_8	8	Resolver	5 / 4	4 / 8
AXIS_9	9	Analog Input	6 / 1	8 / 1
AXIS_10	10	Analog Input	6 / 2	8 / 2
AXIS_11	11	Analog Input	6 / 3	8 / 3
AXIS_12	12	Analog Input	6 / 4	8 / 4
AXIS_13	13	TTL Input	7 / 1	8 / 5
AXIS_14	14	Encoder	13 / 1	8 / 6
AXIS_15	15	Encoder	13 / 2	8 / 7
AXIS_16	16	Encoder	13 / 3	8 / 8
AXIS_17	49	Encoder	9 / 1	
AXIS_18	50	Encoder	9 / 2	
AXIS_19	51	Encoder	9 / 3	
AXIS_20	52	Encoder	9 / 4	
AXIS_21	53	Encoder	10 / 1	
AXIS_22	54	Encoder	10 / 2	
AXIS_23	55	Encoder	10 / 3	
AXIS_24	56	Encoder	10 / 4	
AXIS_25	57	Encoder	11 / 1	
AXIS_26	58	Encoder	11 / 2	
AXIS_27	59	Encoder	11 / 3	
AXIS_28	60	Encoder	11 / 4	
AXIS_29	61	Resolver	12 / 1	
AXIS_30	62	Resolver	12 / 2	
AXIS_31	63	Resolver	12 / 3	
AXIS_32	64	Resolver	12 / 4	

<End of axis table>

Once the axis has been defined, return to the menu bar and press **A** (Axis) **E** (Edit) or <Enter>.

This brings up the list of setup parameters shown in Figure 6-3 that must be entered for each axis used in the application. (It may be necessary to scroll through the list to view all of them.) The setup data is divided into three sections:

1. Scaling data
2. Iterator data
3. Position loop data

There is a default value connected with each parameter. When the selection bar is on a parameter, there is a help window at the bottom of the screen for that parameter. The message offers a brief explanation of the parameter and the range of values that can be entered if it is necessary to change the default value.

The F8 key will bring up a list of choices to select from with some parameters.

The F9 key will provide additional information on the three Input Scaling data lines.

FIGURE 6 - 3. Setup Data Parameter List

1	▶	Main Data		
		SCALING DATA		
		Input	Feedback Units	1
		Scaling	Ladder Units	1
			Ladder Units/Axis Units	1
		Output	Commanded Voltage	10000
		Scaling	Motor RPM at Voltage	1000
			Counts/Motor Revolution	1000
2	▶	ITERATOR DATA		
			Velocity Limit	0.0000000000
			Acceleration Ramp	3839941406.0
			Deceleration Ramp	3839941406.0
			Controlled Stop Ramp	3839941406.0
			Slow Velocity Filter	0
			Fast Velocity Filter	0
			Slow/Fast Velocity Threshold	0.0000000000
			Rollover on Position	No
			Rollover Position	0.0000000000
			Software Upper Limit	536870911.00
			Software Lower Limit	-536870912.0
3	▶	Ignore Limits until Referenced?		Yes
		POSITION LOOP DATA		
			Input Polarity	Positive
			Output Polarity	Positive
			Analog Output Offset	0
			Feed Forward Percent	0
			Proportional Gain	100
			Integral Gain	0
		Plus	Integral Error Limit	0.0000000000
		Minus	Integral Error Limit	0.0000000000
			Derivative Gain	0
			Following Error Limit	0.0000000000
			Update rate	4 millisec
			In Position Band	0.0000000000

AA429-3692

You may find it helpful to scroll through each parameter and read the help window connected with each.

The following tables also describes the parameters.

FIGURE 6 - 4. Descriptions of Scaling Data

Parameter	Description		
SCALING DATA			
<p>Input Scaling - The three lines here establish a scaling ratio between the feedback units the servo software used to perform its calculations and issue its commands, the units used in the ladder, and the units entered in setup.</p> <p>NOTE: When using SERCOS feedback, it is recommended that units be entered in a 1:1 ratio.</p>			
Feedback Units/Ladder Units	<p>Feedback units are the units the servo software uses to perform its calculations and issue its commands. Ladder units are the units used in the ladder program. Ladder units must be integers.</p> <p>The ratio of the feedback device signal to the PiC feedback units is:</p> <p style="padding-left: 40px;">One revolution of a resolver = 4000 feedback units</p> <p style="padding-left: 40px;">One pulse of an encoder = one feedback unit</p> <p style="padding-left: 40px;">NOTE: One cycle of a quadrature type encoder equals four pulses.</p> <p>NOTE: You cannot enter fractional values here.</p>		
Ladder Unit /Axis Units	<p>Axis units are the units of measurement (inches, millimeters, degrees) used in your system. They are used to enter values for several setup parameters.</p> <p>Enter the ratio of ladder units to axis units. (1, 10, 100, 1000, 10000, or 1000000)</p>		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: center;">IMPORTANT</td> </tr> <tr> <td style="padding: 10px;"> <p>Keep in mind that certain iterator and position loop data is entered in axis units. The default values are also in axis units and are sometimes at the upper limit of an acceptable value (ramps and software limits). If you choose a ratio for your ladder units/axis units that is greater than 1, then these default values will exceed the limit. Change the default value to a lower value for those parameters.</p> </td> </tr> </table>		IMPORTANT	<p>Keep in mind that certain iterator and position loop data is entered in axis units. The default values are also in axis units and are sometimes at the upper limit of an acceptable value (ramps and software limits). If you choose a ratio for your ladder units/axis units that is greater than 1, then these default values will exceed the limit. Change the default value to a lower value for those parameters.</p>
IMPORTANT			
<p>Keep in mind that certain iterator and position loop data is entered in axis units. The default values are also in axis units and are sometimes at the upper limit of an acceptable value (ramps and software limits). If you choose a ratio for your ladder units/axis units that is greater than 1, then these default values will exceed the limit. Change the default value to a lower value for those parameters.</p>			

Output Scaling - These three parameters provide the information the controller needs to be able to calculate how much voltage the analog output will need to operate the axis at a certain speed and how many feedback units will be received for each motor revolution.

Commanded Voltage	Enter a commanded voltage in millivolts.
Motor RPM at Voltage	Enter the motor RPM at the commanded voltage entered above.
Counts/Motor Revolution	Enter the number of feedback units that occur for each motor revolution.
	NOTE: These parameters are used for scaling servo calculations by the servo software. They do not represent limits.

Application Note on using PiCTune (see next section) to help determine what scaling information to enter for the number of fu/min/volt of the D/A.

1. Enter 100% for the feed forward value.
2. Move the axis at a fixed velocity in either always the positive or always the negative direction.
3. Observe the following error in PiCTune. If the following error is not zero, the velocity scaling is incorrect.
4. Change the Motor RPM at Voltage in Servo Setup.
For example, if you are jogging in the positive direction and the error is always negative, increase the RPMs and vice versa.
5. Return to PiCTune and perform a field download. Again, observe the following error.
6. Repeat the above steps until the following error is close to zero.
7. Finally, remake the servo setup function and perform a complete download of the ladder.

FIGURE 6 - 5. Descriptions of Iterator Data

Parameter	Description
ITERATOR DATA	
Velocity Limit	Enter the maximum velocity the motor should ever be commanded to in axis units/min. Limits the maximum motor RPM. Does not apply to a slave axis. <div style="border: 3px double black; padding: 10px; margin: 10px auto; width: 80%; text-align: center;"> <p>WARNING</p> <p>The position loop may cause the axis to travel faster than this speed if enough position, integral, and/or derivative error accumulates.</p> </div>

	<p>Typical values are 100,000 to 10,000,000 AU/min (where 1 AU = 1 LU), not to exceed 4095 FU/update. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Acceleration Ramp	<p>Enter the rate in axis units/minute/second at which the servo software will iterate the axis to reach a higher velocity command.</p> <p>Typical values are 10,000 to 10,000,000 AU/min/sec (where 1 AU = 1 LU), not to exceed 1023 FU/update/update. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Deceleration Ramp	<p>Enter the rate in axis units/minute/second at which the servo software will iterate the axis to reach a lower velocity command.</p> <p>Typical values are 10,000 to 10,000,000 AU/min/sec (where 1 AU = 1 LU), not to exceed 1023 FU/update/update. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Controlled Stop Ramp	<p>Enter the rate in axis units/minute/second at which the servo software will iterate the axis to reach a controlled stop.</p> <p>Typical values are 10,000 to 10,000,000 AU/min/sec (where 1 AU = 1 LU), not to exceed 1023 FU/update/update. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Slow Velocity Filter	<p>Enter the milliseconds the filter will take to smooth out a “step” change in velocity while the axis is moving at slow velocities.</p> <p>NOTE: Specifically, the value entered represents the milliseconds that the servo software takes to carry out 63.2% of the step change.</p> <p>Range: 0 - 10000 ms</p>
Fast Velocity Filter	<p>Enter the milliseconds the filter will take to smooth out a “step” change in velocity while the axis is moving at fast velocities.</p> <p>NOTE: Specifically, the value entered represents the milliseconds that the servo software takes to carry out 63.2% of the step change.</p> <p>Range: 0 - 10000 ms</p>
Slow/Fast Velocity Threshold	<p>Enter the velocity in axis units/minute (AU/min) at which the slow or fast velocity filter should be applied.</p> <p>Typical value is 0, not to exceed 4095 FU /update.</p>
Rollover on Position	<p>This parameter acts as a reference reset.</p> <p>Enter “Yes” if you want the axis position to roll over to zero when the Rollover Position entered on the next line is reached.</p> <p>Enter “No” if you do not want the axis position to roll over. The commanded position will then accumulate.</p>
Rollover Position	<p>Enter the rollover position in axis units.</p> <p>Typical values are 100 to 1,000,000 AU (where 1 AU = 1 LU), not to exceed 536870911 FU. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>

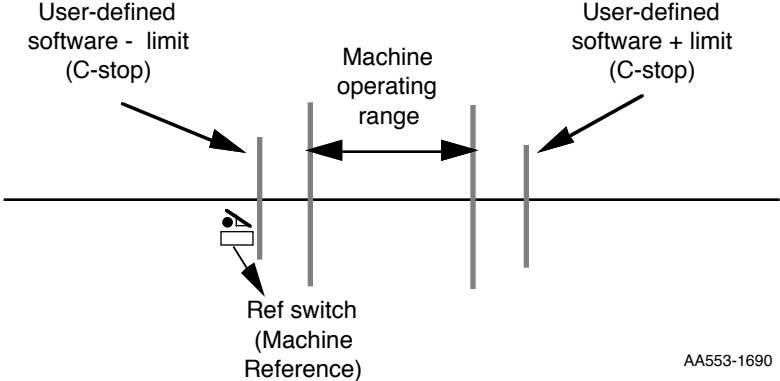
Software Upper Limit	<p>Enter the software-imposed upper travel limit in axis units. Exceeding this limit will generate a C-stop condition.</p> <p>Typical values are 100 to 1,000,000 AU (where 1 AU = 1 LU), not to exceed 536870911 FU. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Software Lower Limit	<p>Enter the software-imposed lower travel limit. Exceeding this limit will generate a C-stop condition.</p> <p>Typical values are 100 to 1,000,000 AU (where 1 AU = 1 LU), not to exceed 536870911 FU. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.</p>
Ignore limits until referenced?	<p>A “Yes” here allows a machine reference to occur without exceeding the software limits designated earlier. This is useful if you exceed the software limits because of how far you have to travel to your reference switch or where your reference switch is located.</p> <p>After the machine reference is complete, the software limits are in effect.</p>  <p>The diagram illustrates the relationship between software limits and a machine reference switch. A horizontal line represents the machine's travel axis. From left to right, there are four vertical markers: 1. 'User-defined software - limit (C-stop)', 2. 'Ref switch (Machine Reference)', 3. 'Machine operating range' (indicated by a double-headed arrow), and 4. 'User-defined software + limit (C-stop)'. The reference switch is located to the left of the machine's operating range, between the software - limit and the start of the operating range.</p> <p style="text-align: right;">AA553-1690</p>

FIGURE 6 - 6. Descriptions of Position Loop Data

Parameter	Description
POSITION LOOP DATA	
Input polarity	<p>Provides a software method of changing the polarity of the position feedback device.</p> <p>If the input polarity is incorrect, the axis will either “run away” or move in the wrong direction. If this occurs, then enter the opposite polarity here.</p>
Output polarity	<p>Provides a software method of changing the polarity of the analog output.</p> <p>If the output polarity is incorrect, the axis will either “run away” or move in the wrong direction. If this occurs, then enter the opposite polarity here.</p>
Analog Output Offset	<p>If it is not possible to get a zero volts reading from a voltmeter placed across the analog output channel for the axis, enter the amount of voltage in millivolts that allows you to reach a zero reading.</p> <p>Range: -10000 to 10000 mV</p>
Feed Forward Percent	<p>Enter a percentage (from 0 to 100%) that you want the position loop to compensate for the lag that occurs between the generation of the following error and the correction of that error by the PID calculations.</p> <p>Range: 0 - 100%</p>
Proportional Gain	<p>Proportional gain calibrates corrective action proportional to the amount of following error. The value entered represents the axis units per minute for each axis unit of following error.</p> <p>Typical values are 1,000 - 5,000 AU/min/AUFE.</p>
Integral Gain	<p>Integral gain determines corrective action proportional to the amount of following error summed over the time duration of the error at a zero velocity command. The longer the following error exists, the greater the integral error. The value entered represents the number of axis units per minute per axis unit of following error times minutes.</p> <p>Typical value is 0. If required, up to 32,000 AU/min/AUFE*min.</p>
Plus Integral Error Limit	<p>Tells the position loop to ignore any additional integral error beyond the value entered when the axis is moving in a positive direction.</p> <p>Typical value is 0. If required, from 100,000 to 500,000,000 AUFE*update (where 1 AU = 1 LU). NOTE: If 1 AU ≠ 1 LU, scale these values accordingly.</p>

Minus Integral Error Limit	Tells the position loop to ignore any additional integral error beyond the value entered when the axis is moving in a negative direction. Typical value is 0. If required, from -500,000,000 to - 100,000 AUFE*update (where 1 AU = 1 LU). NOTE: If 1 AU \neq 1 LU, scale these values accordingly.														
Derivative Gain	Derivative gain determines the corrective action proportional to the magnitude of change of the following error. The value entered represents the number of axis units per min for each axis unit of following error per minute. Typically, this is set to 0. Typical value is 0. If required, up to 500 AU/min/AUFE/min.														
Following Error Limit	Enter the amount of excess error in axis units to allow before an E-stop condition occurs. Typical values are 10 to 10,000 AU (where 1 AU = 1 LU), not to exceed 536870911 FU. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.														
Update Rate	Enter in milliseconds how often the servo upgrade occurs. 4 ms is adequate for most applications. Lower values consume more CPU processing time. If too many axes have too low an update rate, the CPU may not have enough processing time available to run the user program. The iteration rate is eight times the servo upgrade rate: <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>SUG (ms)</th> <th>Iteration Rate (ms)</th> </tr> </thead> <tbody> <tr> <td>.25</td> <td>2</td> </tr> <tr> <td>.5</td> <td>4</td> </tr> <tr> <td>1</td> <td>8</td> </tr> <tr> <td>2</td> <td>16</td> </tr> <tr> <td>4</td> <td>32</td> </tr> <tr> <td>8</td> <td>64</td> </tr> </tbody> </table> For digitizing axes, the update rates are .25, .5, 1, 2, 4, 8, or 16 ms. A rate can be selected for each axis. NOTE: In master/slave moves, the iteration rate is the same as the servo upgrade rate. The update rate for the master axis and the slave axis must be the same in any master/slave move.	SUG (ms)	Iteration Rate (ms)	.25	2	.5	4	1	8	2	16	4	32	8	64
SUG (ms)	Iteration Rate (ms)														
.25	2														
.5	4														
1	8														
2	16														
4	32														
8	64														
In Position Band	Enter the distance in axis units that the axis must be on either side of its endpoint before the in position flag is set. This in position flag can be used in your program. Typical values are 10 to 1,000 AU (where 1 AU = 1 LU), not to exceed 536870911 FU. NOTE: If 1 AU \neq 1 LU, scale these values accordingly.														

Make any necessary changes to the default values for each setup parameter for the first axis in your application.

Other Axes

After entering the setup data for one axis, the setup data for any remaining axes in your application must also be entered.

Leave the setup data screen by pressing <F10>. This accepts all your setup data for AXIS_1.

```

Axis
AXIS_1 IP Slot/Ch:3/1  OP Slot/Ch: 4/1  Encoder/Differential/Quadrature
Main Data
SCALING DATA
Input  Feedback Units 1
Scaling Ladder Units 1
        Ladder Units/Axis Units 1
Output Commanded Voltage 10000
Scaling Motor RPM at Voltage 1000
        Counts/Motor Revolution 1000
ITERATOR DATA
        Velocity Limit 0.0000000000
        Acceleration Ramp 3839941406.0
        Deceleration Ramp 3839941406.0
        Controlled Stop Ramp 3839941406.0
Help Window
Scaling data establishes a scaling ratio between the feedback units the
servo software uses to perform its calculations and issue its commands
and the units of measurement (inches, millimeters, degrees etc.) you use
to refer to your system.
  
```

AA438-3792

The screen where all the axes for your application have been defined appears. In the example below, two axes have been defined. Place the selection bar on AXIS_2 and press <Enter>.

```

Workstation Processor File Axis
Label Axis # Feedback Module Input Slot/Channel Output Slot/Channel
AXIS_1 1 Encoder 3 / 1 4 / 1
AXIS_2 2 Encoder 3 / 2 4 / 2
<End of axis table>
  
```

This brings up the setup data screen again with AXIS_2 at the top. All the setup parameters contain the default values. There are two approaches to changing the data for this second axis.

1. Scroll through the entire list, changing default values to meet the requirements of this axis.
2. Use the copy command to copy setup data from a previously entered axis.

The procedure for using the copy command follows.

Axis

View other axis
Copy other axis

AA434-2690

From the setup data screen for AXIS_2, press **AC** (Axis - Copy).

Copy from axis

Axis Label :

Copy All Fields/
Current Field : CURRENT

Cancel <Esc> Accept <F10>

AA432-2690

This brings up the Copy from axis box with the cursor flashing in the Axis Label field. Enter the label of the axis whose setup data you want to copy. The <F8> key will bring up a list of all axes defined.

Arrow down to the next field, Copy All Fields/Current Field. The default is to CURRENT. This will copy only the field from the setup screen on which the selection bar is placed.

Copy from axis

Axis Label : AXIS_1

Copy All Fields/
Current Field : CURRENT

Copy Field/s
ALL
CURRENT

Cancel <Esc> Accept <F10>

AA433-2690

If you want to copy all fields from the axis, press <F8> to bring up the choices and select ALL.

Copy from axis

Axis Label : AXIS_1

Copy All Fields/
Current Field : ALL

Cancel <Esc> Accept <F10>

AA437-2690

Now all the setup data for AXIS_1 will be copied to AXIS_2. Press <F10>.

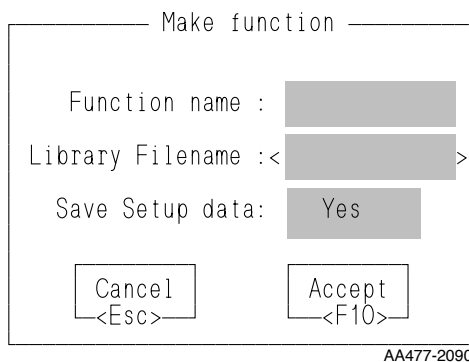
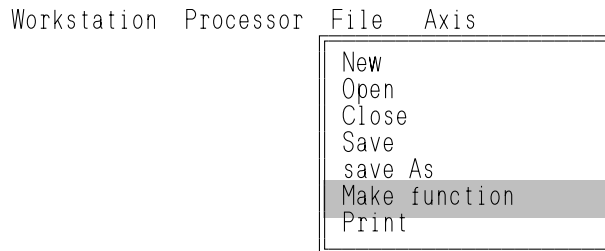
It is still possible to scroll through the setup data for AXIS_2 and change individual parameters if needed. Press <F10> when finished.

Making a Setup Data Function/Saving a .SRV File

Once all the setup data has been entered for all axes used for your application, the next step will be to define the servo function. This function will be stored in the servo library you will create and can then be called into your ladder program to initialize the setup data for your application.

From the File menu, choose Make function block.

Press **FM** (File - Make function)



The Make function box appears and you enter:

1. An appropriate name for the function.
(For example: **USER_SET**)
2. The servo.lib filename you are storing all your servo functions in.
For example: **\PiCLib\Zservo**
(See the section below on creating a library for user-defined functions.)
A .lib extension is added automatically.
3. The Save Setup data line defaults to “Yes.” This ensures that all your setup data will be saved in the .SRV file.

Creating a Servo Library for User-Defined Functions

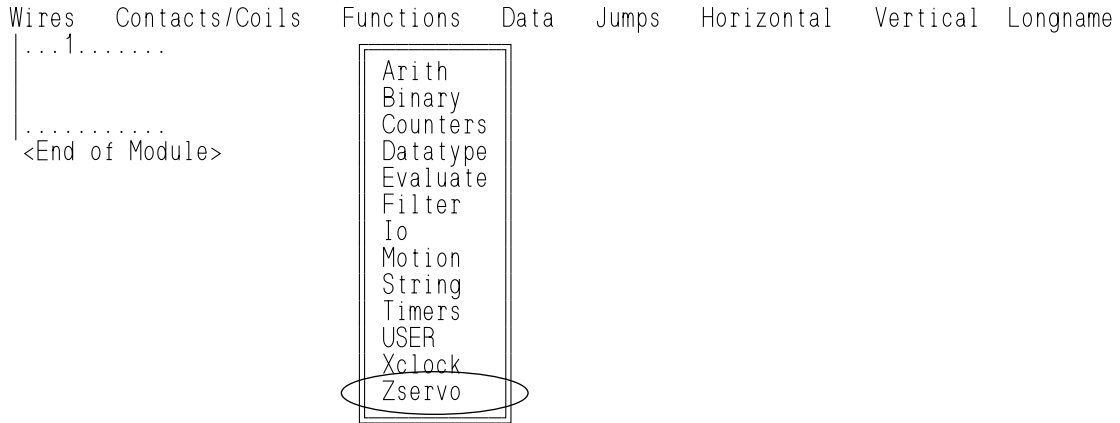
NOTE: It is recommended that you create a separate library for all user-defined functions. This will keep them separated from the standard functions found in the PiCLib directory.

As an example, choose a name for your library which will have a unique first letter (or hot key) so that when it appears in the Functions menu of PiCPro with the standard functions and function blocks, it will be easy for you to access while programming. A name like “Zservo” will place it at the end of the Functions list as shown in Figure 6-4.

Create a path to your user-created library and enter that path in Library Filename in the Make function box. In this example, **\PiCLIB\ZSERVO** could be typed. This would put Zservo in the PiCLib directory.

You can also use the Zservo library to hold any profile functions you define for master/slave applications. These are covered in the next chapter.

FIGURE 6 - 7. User-Defined Function Blocks in PiCPro Functions Menu.

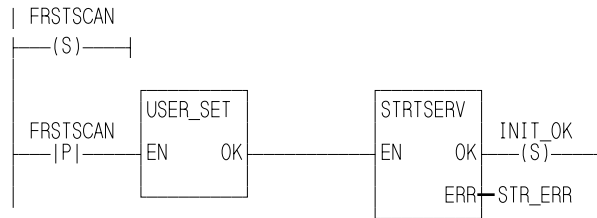


Using **Z** as a hot key will bring up a list of all the functions you have defined and stored in the Zservo library.

Initialize setup data

The servo setup function you make with the Servo setup program containing the setup data for all axes used in your application must be incorporated into your ladder program. To initialize all the setup data you use the standard motion function STRSERV with your setup function. Figure 6-8 below illustrates a network designed in PiCPro that does this.

FIGURE 6 - 8. Initializing Setup Data in Your LDO



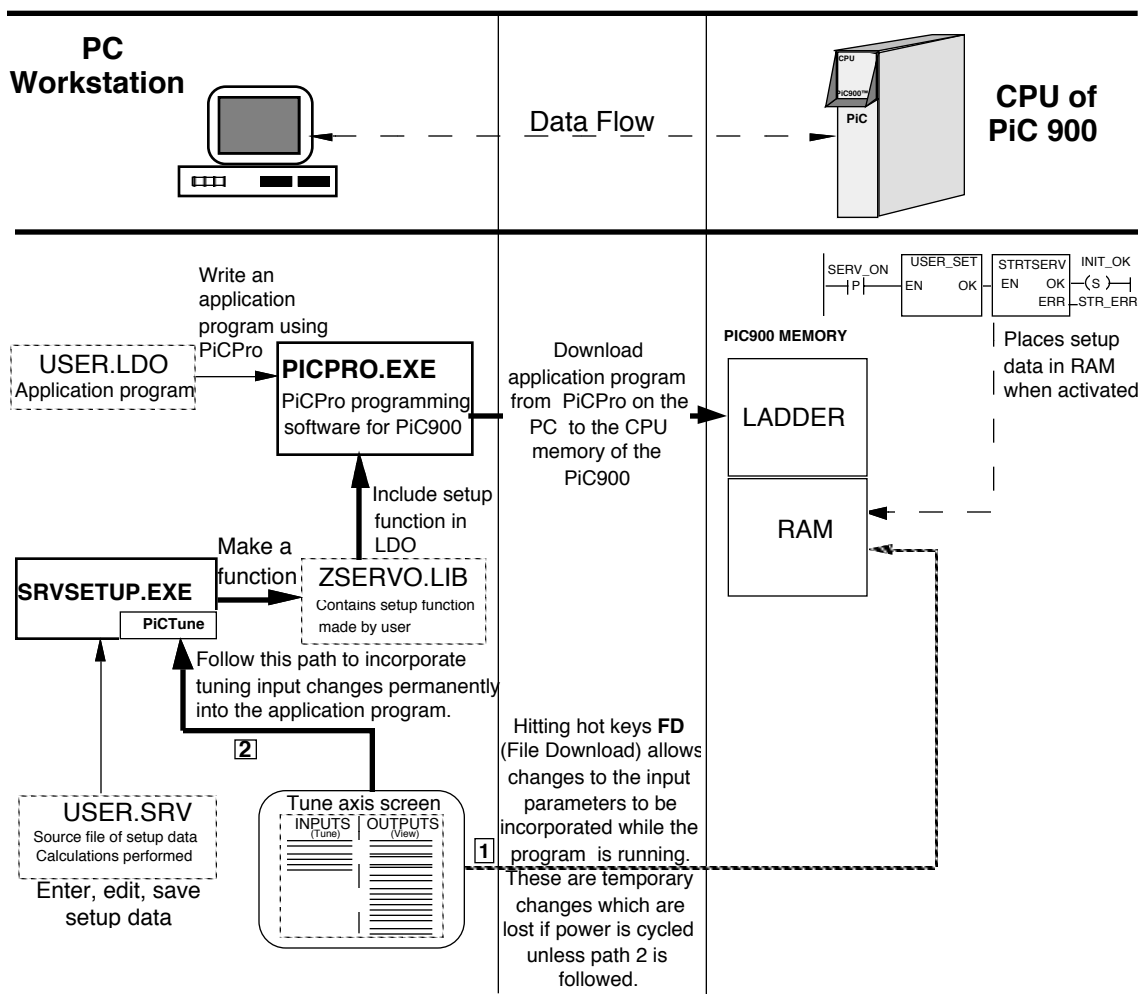
Introduction to Tuning

The tuning portion of the Servo setup and tuning (SRVSETUP) program provides remote tuning capabilities. Adjustments to offsets and gains can be done through software instead of manual potentiometer settings on the hardware modules.

Figure 6-9 illustrates how PiCTune works with the PiC. Your program would be downloaded and running on the PiC. You bring up the PiCTune screen on the PC and make any changes to the inputs.

You can download those changes directly from the PiCTune screen (path 1). If you want the changes to be a permanent part of your setup parameters, then leave the PiCTune screen and return to the axis table screen and make a function (path 2).

FIGURE 6 - 9. PiCTune Data Flow



Key Represents data files created by user Represents executable files supplied with PiCServoPro

AA577-4992

Axis Tuning

The servo axis tuning feature, available under the Axis menu of Servo Setup (Tune axis) or <Control T> allows you to view the Outputs column and change the Inputs column on the screen shown in Figure 6-10 for the designated servo axis.

FIGURE 6 - 10. PiCTune Screen

Axis	Processor	Fields	Inputs	Outputs
			AXIS_1 Inputs	AXIS_1 Outputs
▶	Proportional Gain	100		Proportional Gain 101
▶	Integral Gain	0		Integral Gain 0
▶	Derivative Gain	0		Derivative Gain 0
▶	Analog Output Offset	0		Analog Output Offset 0
▶	Slow Speed Filter	0		Slow Speed Filter 0
▶	Feed Forward Percent	0		Feed Forward Percent 0
				Following Error 0
				Commanded Velocity 0
				Actual Position 0
				Commanded Position 0
				Analog output in mv 0
				Active Move None [000]
				Next Move None [000]
				EStop Error
				CStop Error
				Prog Error Low Byte
				Prog Error High Byte
				Servo Loop State Open

NOTE

The items listed in the Inputs column can also be written to and read from your ladder using the TUNERead and TUNEWRIT functions.

NOTE

Pressing the <F9> Help key when in the PiCTune screen will bring up a description of all the E, C, and P errors. Use the <PgUp> and <PgDn> keys to scroll through the categories.

In PiCTune you can view two outputs in the Outputs column for a digitizing axis as shown below.

Axis	Processor	Fields	Inputs	Outputs
			1 Inputs	1 Outputs
▶	No Inputs			Actual Position 0
				EStop Error

NOTE: The only Estop error that appears with a digitizing axis is the loss of feedback error.

Changing Inputs

There are six parameters in the input column that you can change. The procedure to change an input is as follows:

1. Place the selection bar on the parameter you want to change.
2. Enter the new value.
3. Be sure the parameter is marked. It is marked if a triangle is on the left of the parameter as shown in Figure 6-11. When you enter PiCTune, the default is to have all six parameters marked.

FIGURE 6 - 11. Marked Input Parameters in PiCTune

Axis	Processor	Fields
		AXIS_1 Inputs
	▶	Proportional Gain 100
	▶	Integral Gain 0
	▶	Derivative Gain 0
	▶	Analog Output Offset 0
	▶	Slow Speed Filter 0
	▶	Feed Forward Percent 0

Marked inputs — [bracket pointing to the left of the parameter list]

The parameter can be unmarked by pressing **FM** (Fields - Mark/Unmark) with the selection bar on the parameter. See Figure 6.12.

FIGURE 6 - 12. Mark/Unmark Toggle

Axis	Processor	Fields
		AXIS_1 Inputs
	▶	Proportional Gain 100
	▶	Integral Gain 0
	▶	Derivative Gain 0
	▶	Analog Output Offset 0
	▶	Slow Speed Filter 0
	▶	Feed Forward Percent 0

Mark/Unmark
Download

Any parameter that is unmarked will not be downloaded to the PiC. In Figure 6-13, the first three parameters are unmarked and the last three are marked.

FIGURE 6 - 13. Marked and Unmarked Parameters

Axis	Processor	Fields
		AXIS_1 Inputs
Unmarked inputs		Proportional Gain 100
		Integral Gain 0
		Derivative Gain 0
Marked inputs	▶	Analog Output Offset 0
	▶	Slow Speed Filter 0
	▶	Feed Forward Percent 0

4. After entering all values you want to change, press **FD** (Fields - Download). The values you entered will take effect immediately in your ladder program. They will remain in effect until you either change and download them again or the SRTSERV function in your program is reinitialized.
5. To tune or view another axis in your application, press **AV** (Axes - View). Type in the label of the next axis you want to tune or view (or press <F8> and select the axis from the list). Press <F10>. The axis label you chose now appears at the top of the PiCTune screen. Proceed as described above.

6. If you want your changes to become a permanent part of your program, leave PiCTune by pressing <F10> and make a function to replace the previous one. *Download your ladder program again.* Now the changes you made in PiCTune are permanently in your ladder program.
7. To leave the PiCTune screen, press <F10>. Pressing <Esc> allows you to leave the PiCTune screen without saving any changes in the workstation. Any changes that have been downloaded to the PiC are in RAM memory of the PiC until the servos are reinitialized.

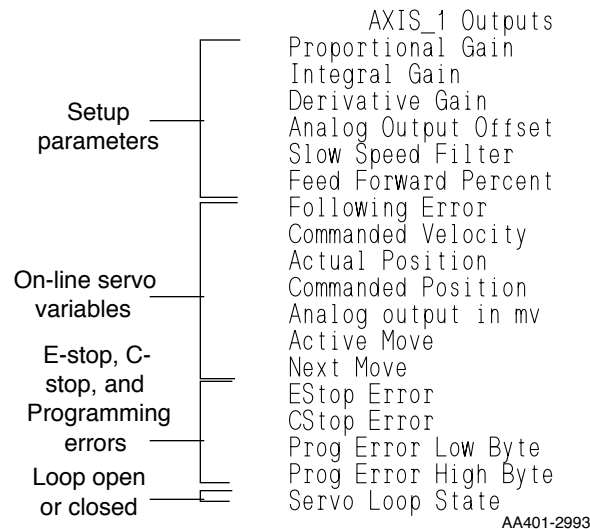
The values you have changed in PiCTune appear with the specific parameter in the Edit axis data screen.

Viewing Outputs

The outputs on the right side of the PiCTune screen can only be read, not written. They are divided into four groups as shown in Figure 6-14.

1. Setup parameters
2. On-line servo variables.
3. Errors
4. Loop status

FIGURE 6 - 14. Outputs on PiCTune screen



Setup parameters

The first six outputs are setup parameters from the Edit axis data screen--the same six that can be written to in the input column of the PiCTune screen.

On-line servo variables

The next seven outputs are on-line servo variables. They are being read back to you as they occur.

Errors

The E-stop, C-stop, and programming errors can be viewed from the PiCTune screen. The ESTOP Error and the CSTOP Error each have eight bits displayed. The Prog Error Low Byte and the Prog Error High Byte display 16 bits for programming errors.

An error is present if an “E” appears in any bit location. These errors can be identified by pressing the <F9> Help key or by referring to the tables found in Appendix C or the descriptions of the E_ERRORS, C_ERRORS, and P_ERRORS functions.

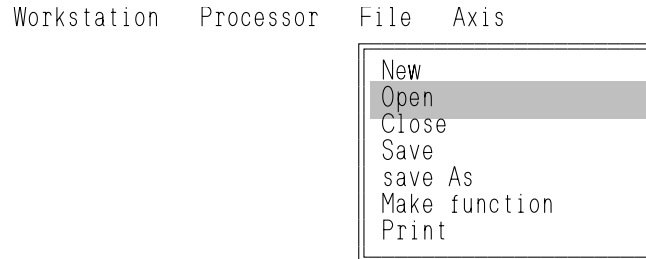
Loop status

This output indicates if the D/A is zeroed (open loop) or is attempting to zero the following error (closed loop).

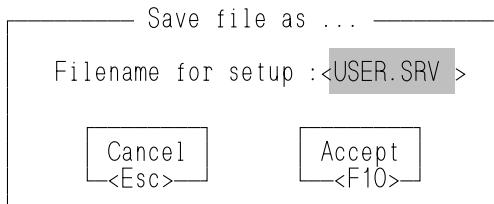
Other Features of the Servo Setup Program

File Menu of Servo Setup

The New, Open, and Close commands allow you to create a new .SRV file, open an existing one, or close the one you are working on from within the Servo setup program.



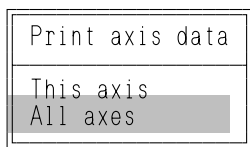
Save creates a *your name*.SRV file which contains all the setup data you entered with the Servo setup program. It performs the same function as choosing “Yes” for the Save Setup data in the Make Function box.



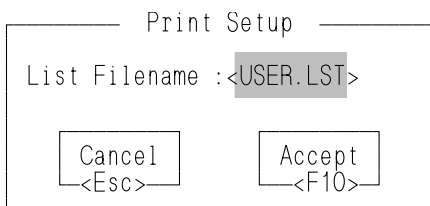
AA552-1690

If save As is chosen, the Save file as box appears and you can type in a new filename over the old. The .SRV extension is added automatically.

This is a copy function. Both the file with the old name and the file with the new name will be in your directory.



AA405-2390



AA404-2390

The Print command creates a text (.LST) file of your setup data on disk. This can then be sent to your printer.

You can also go directly to the printer by entering the printer port name in the Print Setup box (i.e., LPT1). No .lst file is created.

You have the option of printing the setup data for the axis you have the selection bar on or all the axes in your setup file. Make your selection.

The Print Setup box appears with the name of the setup file and the .LST extension.

The setup of your printer is handled with the Printer setup command found under the Workstation menu.

Other Features of the Axis Menu

Setup Table Screen

Other features under the Axis menu in the table screen give you the ability to redefine an existing axis with the Redefine axis command. This changes the hardware definition of the axis. The axis label can also be changed here. Be sure to make a function again and download your program to incorporate the new definition.

You can delete an axis no longer needed with the Delete command.

Setup Edit Screen

Another feature under the Axis menu in the setup edit screen is the VIEW command which allows you to view the setup parameters for a second axis.

PiCTune Screen

The View other axis command allows you to view a second axis on the same screen. Enter the label of the second axis (or use the <F8> key) in the View Axis box and press <F10>. The PiCTune screen now shows tune information for both axes with the first one active.

The axis menu adds a Resume command which toggles back to displaying the first axis only.

CHAPTER 7 Axis Profile Setup

When using the PiC in a master/slave application where the motion of a servo axis (slave) is tied to the position transducer of another axis (master), position or *ratio* profiles using the PiC Profile program can be defined. Each profile allows the slave axis to respond in a multitude of accelerations, decelerations, and velocities to the position of the master axis during a single cycle.

There can be from 1 to 18 ratio profiles initialized in the LDO. These profiles are used with the RATIOPRO function.

Axis profile setup (PiC Profile) is a program development tool which allows you to do this. This chapter provides some background information on profiles and explains how to use the PiC Profile program.

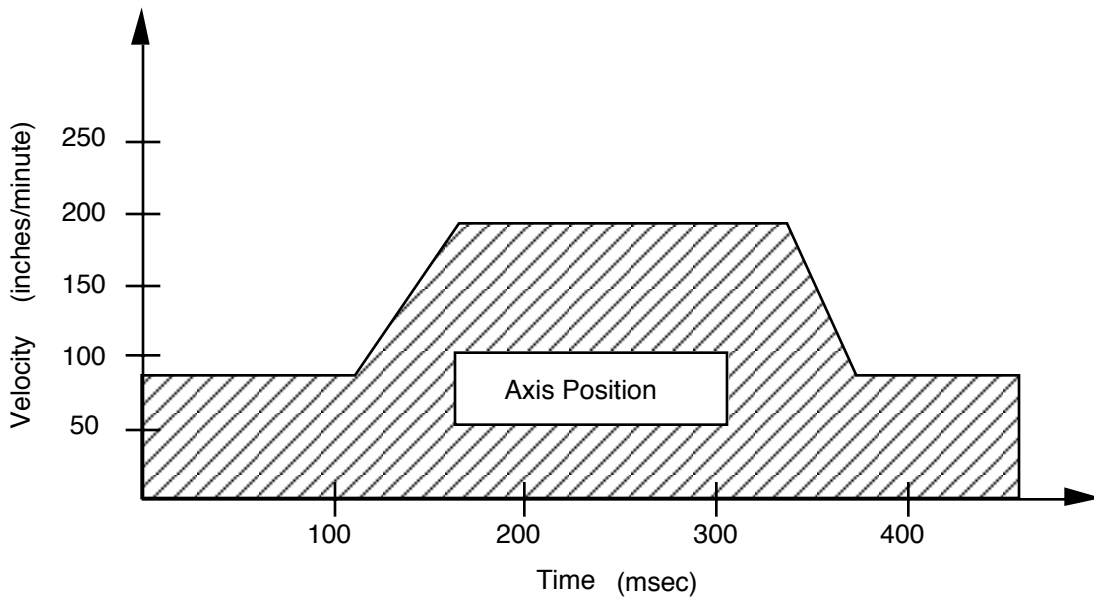
Background Information on Ratio Profiles

With absolute positioning, incremental distance, and velocity moves, motion is based on acceleration, velocity, and distance.

With master/slave moves, motion is based on the position of the master axis.

In many servo applications, velocity profiles are used to generate positioning sequences for the servos as illustrated in Figure 7-1. The horizontal coordinate represents elapsed time and the vertical plane represents velocity. The area under the velocity curve is the axis accumulated distance. At any point in time, the axis position can be calculated.

FIGURE 7 - 1. Velocity Profile

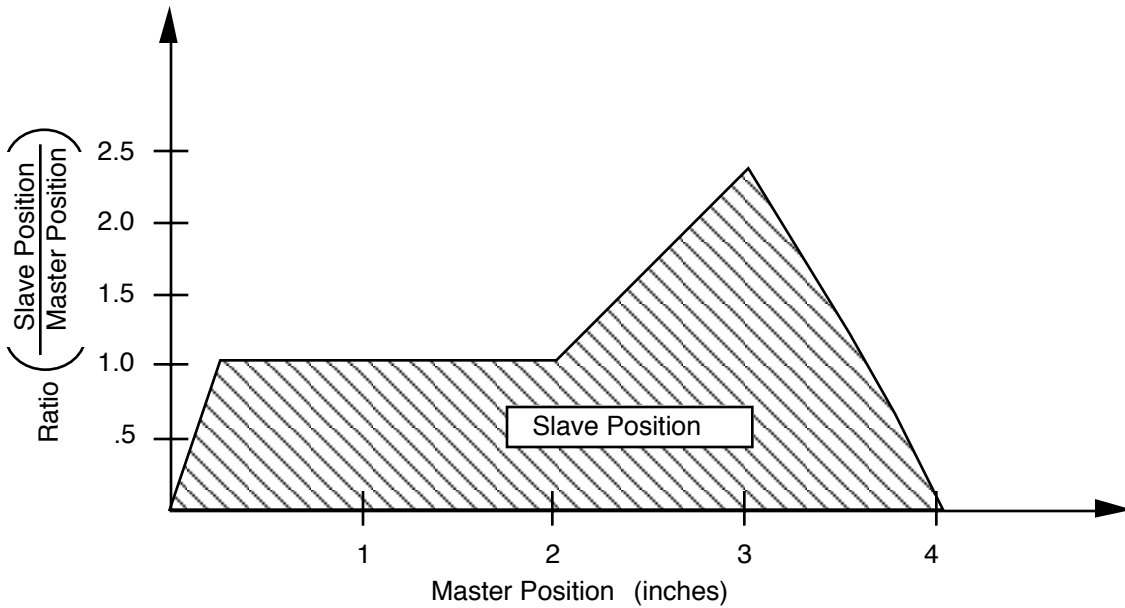


$$\text{Velocity} \times \text{Time} = \left(\frac{\text{Position}}{\text{Time}} \right) \times \text{Time} = \text{Axis Position}$$

AA588-2290

The PiC uses ratio profiles to generate positioning sequences for servos. In synchronization applications where an axis is slaved to a master position transducer, ratio profiles are independent of time. In the ratio profile shown in Figure 7-2, the horizontal coordinate is the master accumulated distance and the vertical plane is the ratio of the slave distance to the master distance. Now the area under the curve is the slave accumulated distance and is tied to the master position, not time.

FIGURE 7 - 2. Ratio Profile



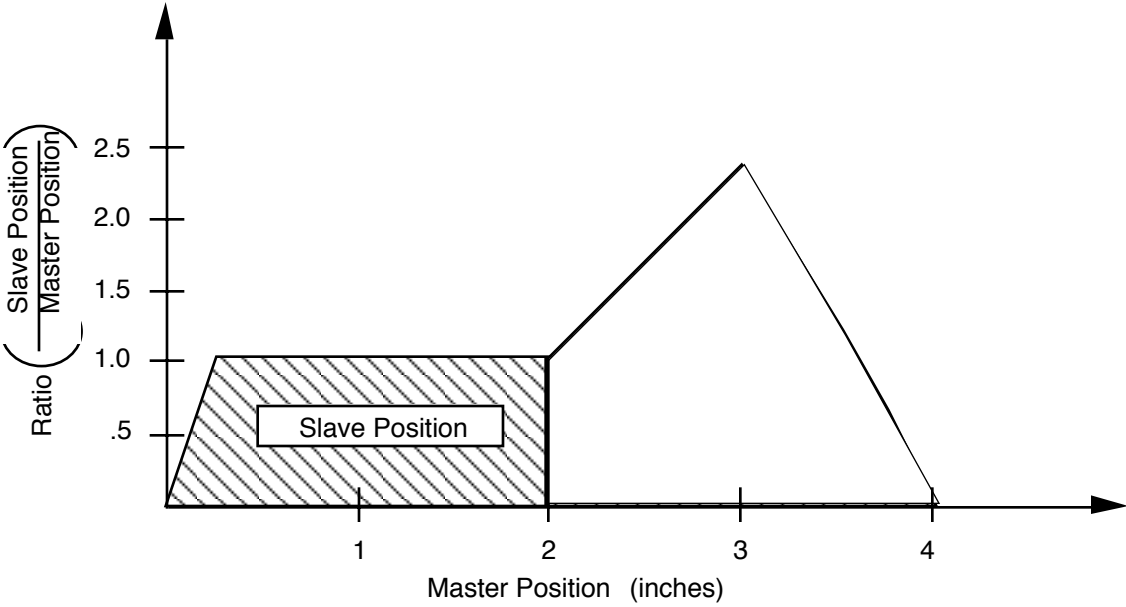
$$\left(\frac{\text{Slave position}}{\text{Master position}} \right) \times \text{Master position} = \text{Slave position}$$

AA589-2290

The profile defines where the slave should be in relationship to where the master is at any point in the cycle. The servo software uses this information to calculate the position command it sends to the position loop for controlling the slave axis.

The position of the slave at any given point is dependent on where the master is. It is independent of time. In Figure 7-3, when the master reaches 2 inches, the slave would have covered the shaded area under the curve. It would not matter if it took the master 1 minute or 1 hour to travel 2 inches.

FIGURE 7 - 3. Slave Position After Two Inches of Master Travel



$$\left(\frac{\text{Slave position}}{\text{Master position}} \right) \times \text{Master position} = \text{Slave position}$$

AA590-2290

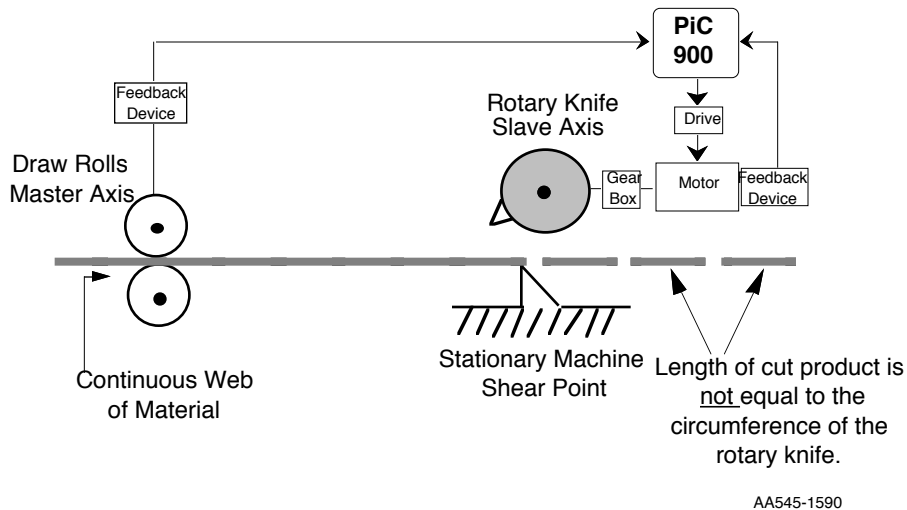
This provides a very easy method of programming the speed of an operation. If you want to increase the speed of the cycle, adjust the speed of the master axis; the slave axis will follow automatically.

An Example of Using a Ratio Profile

One example for using a ratio profile is shown in Figure 7-4. This application represents a cutting operation. Some features to note include:

- The draw rolls represent the master axis. It is not always necessary to have the PiC controlling the master axis. The PiC may just be receiving feedback information from the master axis as illustrated in Figure 7-4.
- The rotary knife represents the slave axis. This axis is controlled by the PiC.
- Because the length of the cut product does not equal the circumference of the rotary knife, the slave and master axes cannot be operating at a constant ratio throughout each cycle. The ratio must be constant during the cut. Then it will be necessary for the slave to accelerate and decelerate in order to be in position for the next cut.

FIGURE 7 - 4. Ratio Profile Cutting Operation

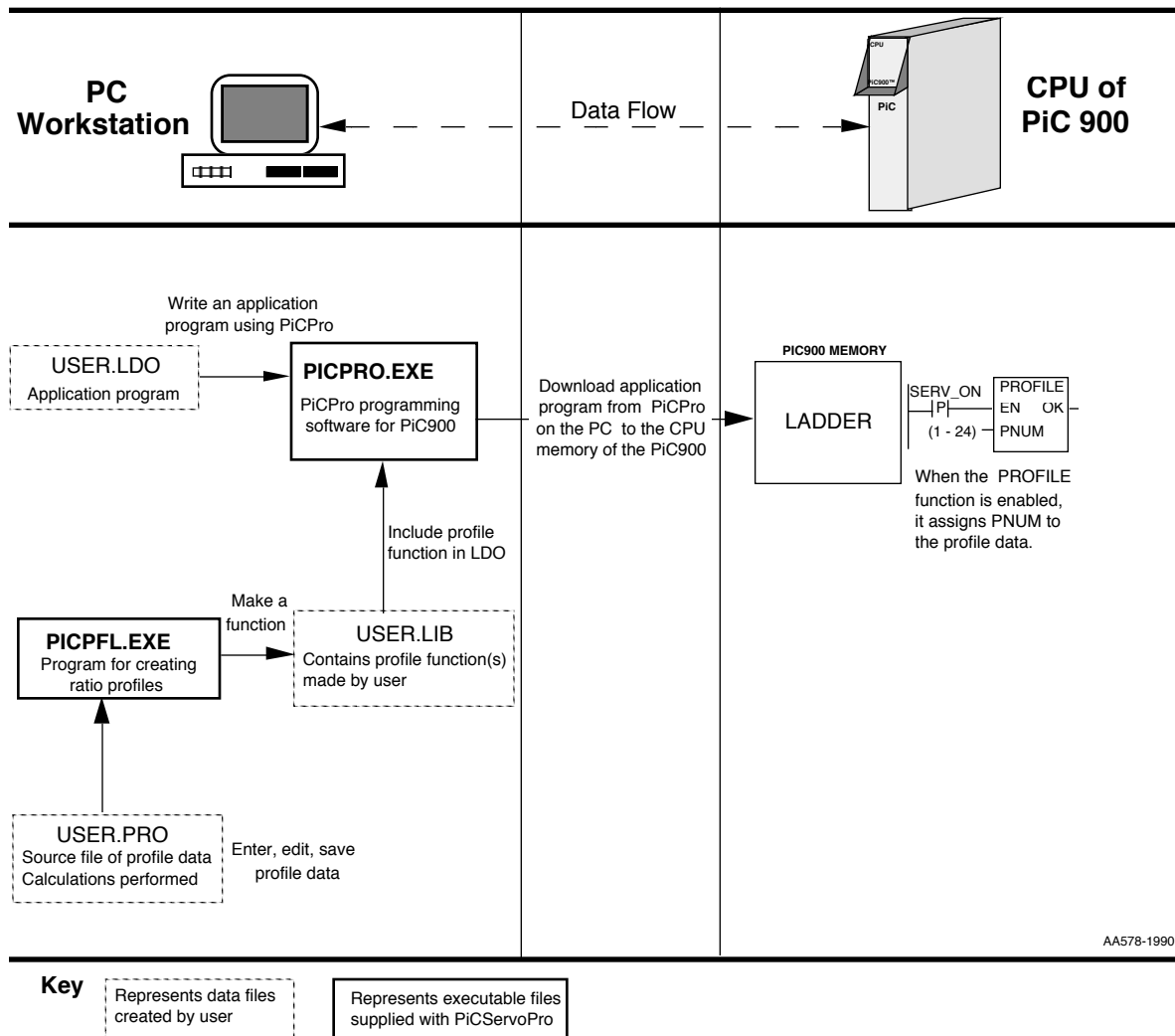


Introduction to the PiC Profile program

The PiC Profile program of PiCServoPro provides the capability of defining profiles that can be used in master/slave applications. Figure 7-5 illustrates how the PiC Profile program works with the PiC. It is similar to the Servo setup program.

You enter the data to define a profile in a .pro file, make a function for the profile data, incorporate that function in your ladder program (.LDO file), and download the program to the PiC. When the profile function is called in the ladder, the data is initialized and you are ready to run the profile with a RATIOPRO standard motion function.

FIGURE 7 - 5. Data Flow for the PiC Profile Program



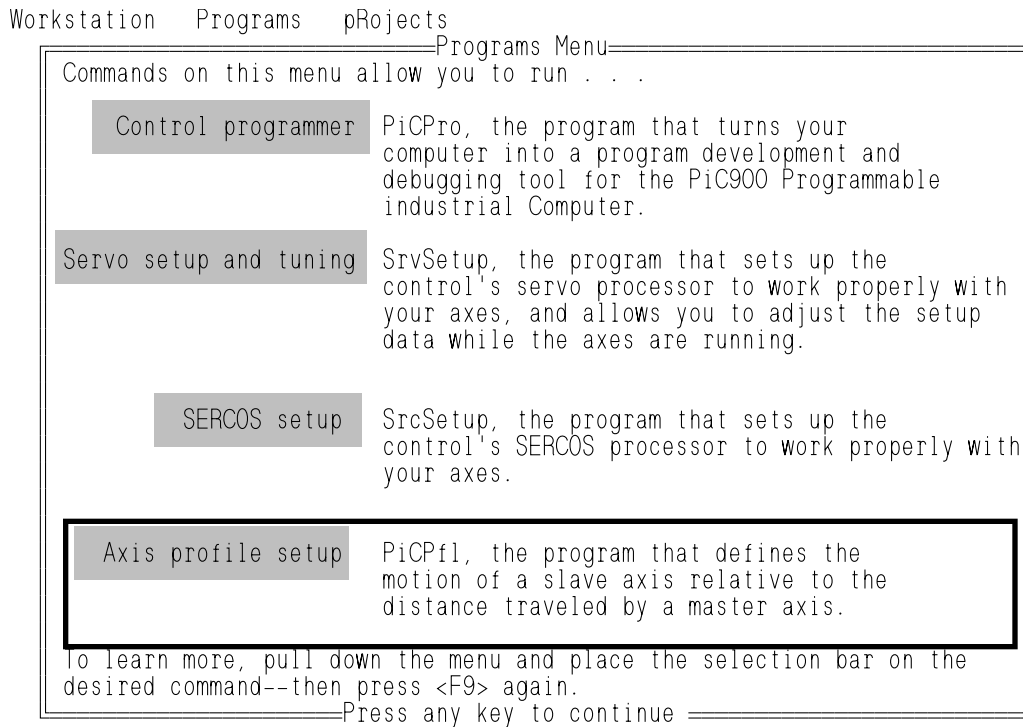
Entering Profile Data

IMPORTANT

The ratio used in any profile should not cause the slave axis to exceed the mechanical limits of its maximum velocity rate or maximum acceleration rate.

With PiCServoPro running on the PC, the help message (<F9>) when the selection bar is on Programs explains what the PiC Profile program allows you to do (Figure 7-6).

FIGURE 7 - 6. Axis Profile Setup from Programs Help Message



NOTE

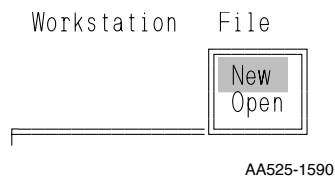
When moving through the PiC Profile program, press <F9> for a help message on any item the selection bar is placed on.

Press **P** to display the Programs menu.



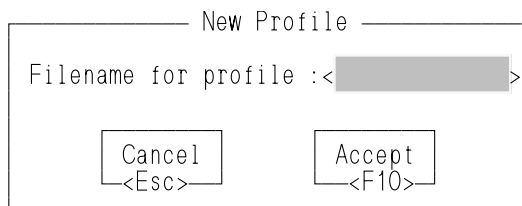
With the Programs menu displayed, press **A** to select the axis profile setup (PiCPfl) program.

The menu bar changes to Workstation File as shown on the left.



Press **FN** (File - New) to create a new file for the profile you will be defining.

[Press **FO** (File - Open) to open files containing profiles that have already been defined. Remember that <F8> will provide a list of previously created files.



Type in a filename for the profile you are about to define and press <F10>. (If you are opening a previously created file, that filename should appear in the Open Profile File box.)

Press <F10>.

Define Units

		MASTER AXIS DATA	
Input	Feedback Units:		1
Scaling	Ladder Units:		1
F8-Choices	Ladder Units/Axis Unit :		1
		SLAVE AXIS DATA	
Input	Feedback Units:		1
Scaling	Ladder Units:		1
F8-Choices	Ladder Units/Axis Unit :		1

AA528-1590

The Define Units box appears. Here you enter the number of feedback units from the feedback device per the number of ladder units and the ladder unit/axis unit value for the master and slave axes. A scaling factor is created by this data allowing the workstation to convert axis units entered by you to feedback units used to perform calculations.

NOTE: If you use axis units to enter profile data, the servo software may have to round off values when converting the axis units to feedback units for calculations. In some applications, this could cause inaccurate positioning.

To avoid this, use feedback units to enter profile data. Do this by having feedback units equal ladder units and the ladder units/axis units equal to "1" in the Define Units box. Now you will be entering profile data in feedback units.

It is recommended that the units you define for setup data match what you enter here for profile data.

The screen below appears with the master and slave axis data you entered displayed. The menu bar changes to Workstation File Segment.

From this screen, you will enter all the segments for the profile you are creating. Note that *your name.pro* appears at the bottom of the screen


```

Workstation  File  Segment
Master Axis  Slave Axis  Selection Bar on Segment:1
Feedback Units: 50000  40000  Total Master Distance : 0
Ladder Units: 50000  40000  Total Slave Distance : 0
LU / AU : 1 1
  
```

```

.....To add a segment, select Insert from the Segment menu
<End of Profile Segments>
  
```

AA529-1590

Press **SI** (Segment - Insert) to bring up the Insert segment box. Enter two of any of the three variables listed in the box. The third will be calculated for you.

IMPORTANT

The PiC Profile program ensures that the starting ratio of the first segment is "0." You must ensure that the ending ratio of the last segment is "0."

```

Workstation  File  Segment
Master Axis  Slave Axis  Selection Bar on Segment:1
Feedback Units: 50000  40000  Total Master Distance : 0
Ladder Units: 50000  40000  Total Slave Distance : 0
LU / AU : 1 1
  
```

.....to add a segment menu

<End of Profile Segme

Insert segment

Master Distance : < >

Slave Distance : < >

Ending Ratio : < >

Cancel
<Esc>

Accept
<F10>

An example profile with five segments is shown below. The total distance of the master and slave are shown in the upper right. The Ending Ratio column shows the ratio at the end of that segment.

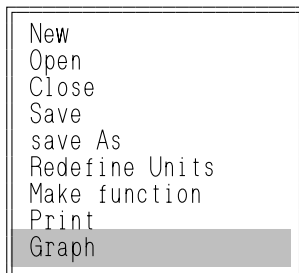
The formula used to calculate the ending ratio of each segment is:

$$\text{Ratio}_{\text{end}} = 2 \left(\frac{\text{slave distance}}{\text{master distance}} \right) - \text{Ratio}_{\text{start}}$$

Workstation	File	Segment	Master Axis	Slave Axis	Selection Bar on Segment: 1
Feedback Units:			50000	40000	Total Master Distance : 14
Ladder Units:			50000	40000	Total Slave Distance : 13
LU / AU :			1	1	
		Master Distance	Slave Distance	Ending Ratio	
		4	2	1	
		4	4	1	
		2	3	2	
		2	3	1	
		2	1	0	

AA440-1590

Workstation File Segment



AA531-1590

The PiC Profile program will plot a graph of your profile if your workstation is equipped with a CGA, EGA, VGA, or a Hercules Hi-Resolution Monochrome graphics card. See note below.

To view the graph, press **FG** (File - Graph).

NOTE ON PANNING PROFILES

When panning a profile wider than your screen, the new screen may overwrite the old instead of erasing it. This problem usually occurs on workstations with older Hercules Hi-Res Monochrome or CGA cards or video cards that are not completely compatible with the IBM standard.

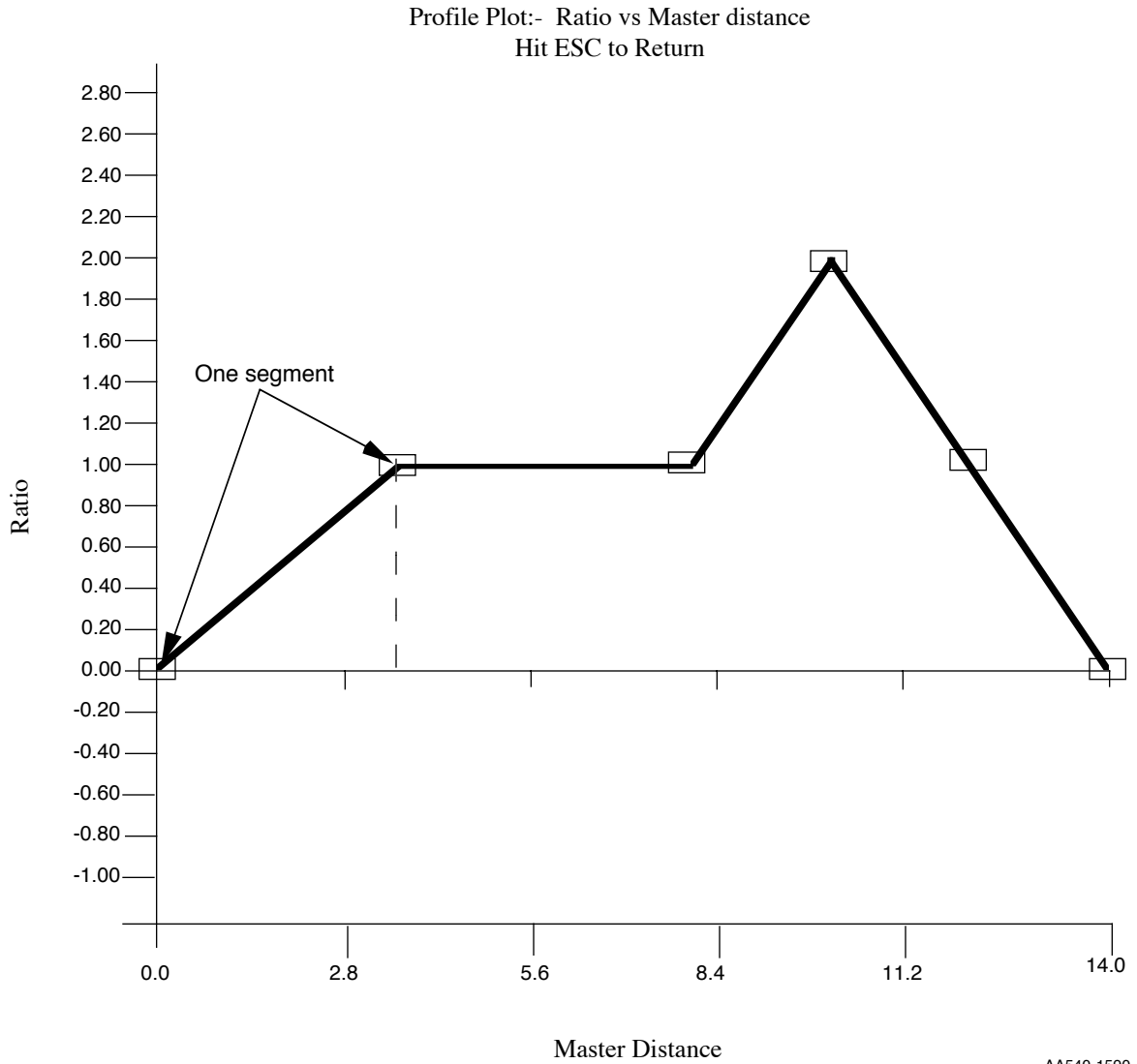
The problem can be corrected by installing the ANSI.SYS in your CONFIG.SYS file using the following:

DEVICE = C:\DOS\ANSI.SYS

This assumes that ANSI.SYS is in your DOS directory on the C drive. Substitute the path where your ANSI.SYS is for C:\DOS\. See your DOS manual for additional information on installing DEVICE drivers.

The graph in Figure 7-7 illustrates the plot of a profile with five segments. A segment is represented by the area between two consecutive rectangular markers.

FIGURE 7 - 7. Plot of Ratio Profile



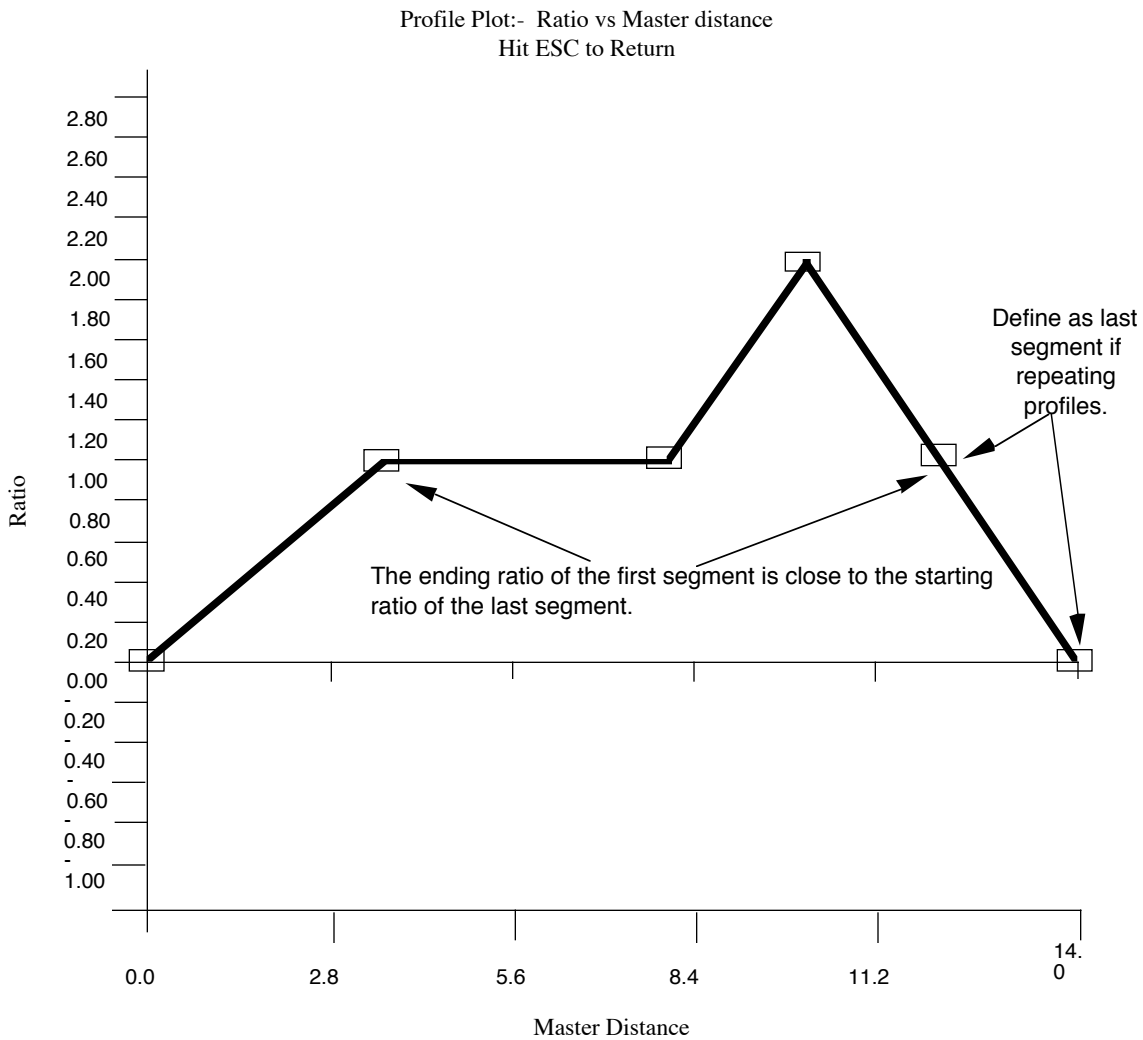
STARTING
PiCPro/PiCServoPro

AA540-1590

A new segment usually begins where the slope of the ratio between the slave and the master changes. However, as shown in Figure 7-8, the beginning of the last segment does not represent a change in the slope. It was defined as a separate segment because the profile will be repeated.

Whenever a repeating profile is used, it is important to keep the ending ratio of the first segment close to the starting ratio of the last segment.

FIGURE 7 - 8. First and Last Segments of Repeating Profiles

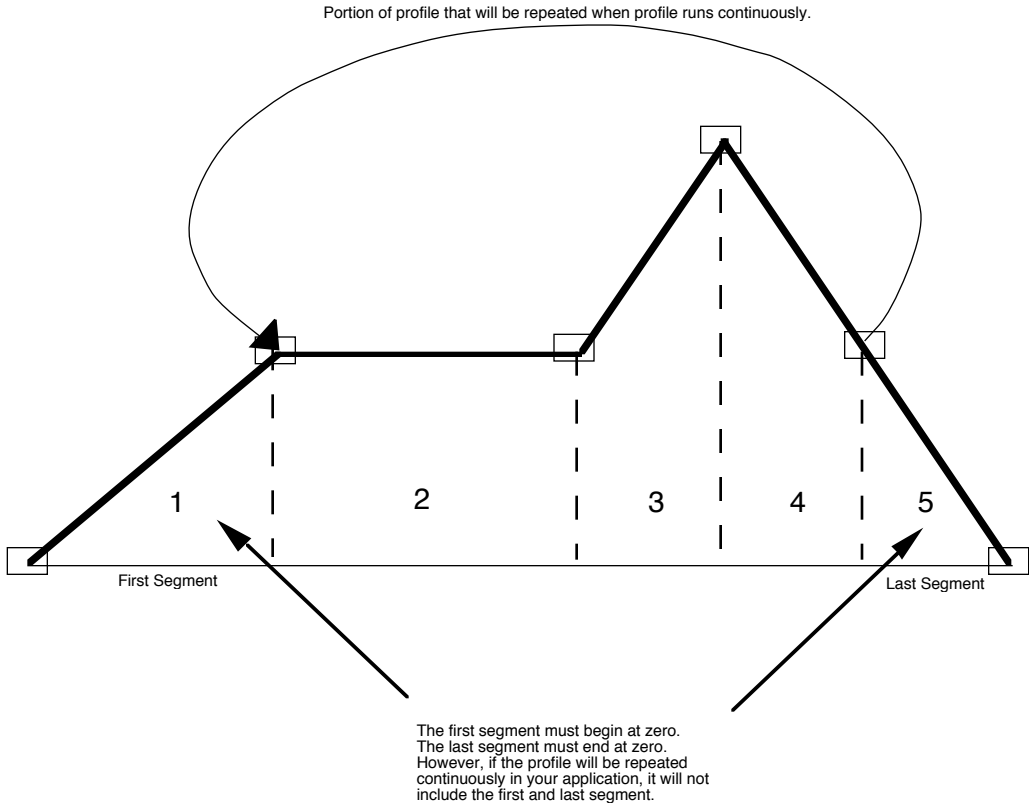


These ratios should be matched as close as possible so that there is not a large “step” for the slave axis once repeating is started. The first and last segment of a repeating profile are dropped as illustrated in Figure 7-9.

NOTE

The first segment of a profile is the acceleration period for the move. The last segment of the profile is the deceleration to stop for the move. Both these segments are ignored during continuous operation.

FIGURE 7 - 9. Repeating Profile

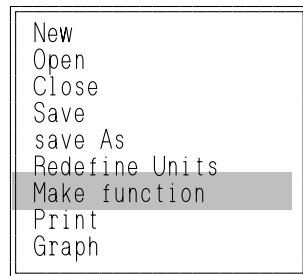


Making a Profile Function/Saving a .PRO File

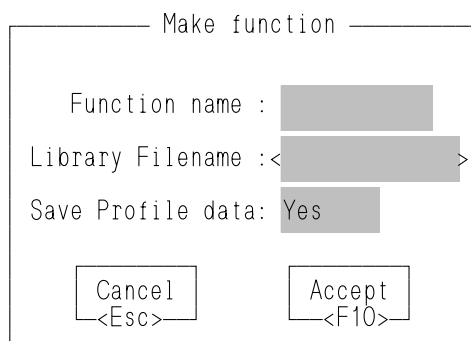
Once the profile has been defined, the next step is to make a function. This function will be stored in the same servo library you created to hold the setup function and can then be called into your ladder program as required.

Workstation File Segment

From the File menu, choose Make function. Press **FM**.



AA536-1590



AA593-2390

The Make function box appears and you enter:

1. An appropriate name for the function. (For example, PROF_1)
2. The servo.lib filename you are storing all your servo functions in (/PiCLib/Zservo).
3. “Yes” to save all the profile data for this profile in a .pro file. (The default is “Yes.”)

Press <F10> to accept data.

Using the Profile Function

You have now completed the following tasks:

1. Defining a ratio profile using the PiC Profile program
2. Creating a profile function
3. Saving the profile data in a .pro file

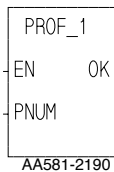
You may have made one or more profile functions. Each function can be called in your ladder program as needed.

IMPORTANT

Be sure to follow the two steps listed below *in the order given* when using profiles:

1. Initialize the servo data.
2. Initialize the profile data by including the profile function in your LDO *before* calling the RATIOPRO function that uses it.

The profile function will look like the one shown below.

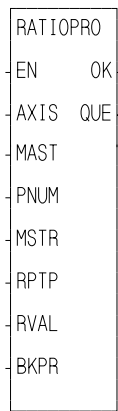


This is the profile function defined when making a function. It must be enabled in your ladder program in order to initialize the axis profile data you entered using the PiC Profile program.

Inputs: EN (BOOL) = enables execution

PNUM = (USINT) The number you assign to the profile (range from 1 to 18) giving it a specific memory location separate from any other profiles you may have created for your program.

Outputs: OK (BOOL) = execution complete (The OK will not be set if a number other than 1 through 18 has been entered in PNUM.)



AA569-5190

Once the axis profile setup data has been initialized, a RATIOPRO function is entered in your LDO in order to begin execution of the ratio profile identified at PNUM.

- Inputs:**
- EN (BOOL) = enables execution
 - AXIS (USINT) = identifies slave axis
 - MAST (SINT) = master axis
 - PNUM (USINT) = profile number (1 through 18)
 - MSTR (DINT) = Master start position
 - RPTP (BOOL) = repeat profile
 - RVAL (BOOL) = reversal allowed
 - BKPR = (BOOL) = back to back profiles

Outputs: OK (BOOL) = execution complete

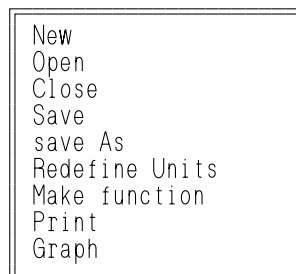
Other Features of the PiC Profile Program

This section explains other items available with the PiC Profile program.

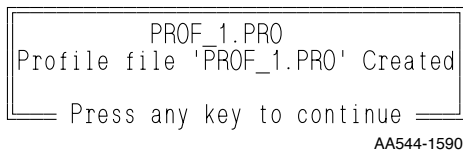
File menu

Under the file menu, the New, Open, and Close commands allow you to define a new profile, open an existing .pro file (use the <F8> key to display a list of existing profiles), and close a file while in the PiC Profile program.

Workstation File Segment

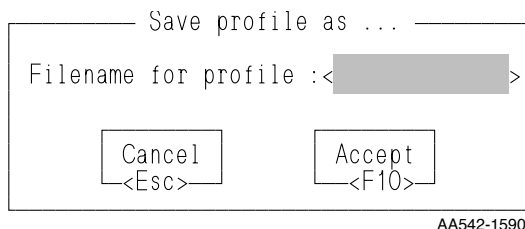


AA531-1590



The Save command creates a .pro file containing the profile information you entered with the PiC Profile program.

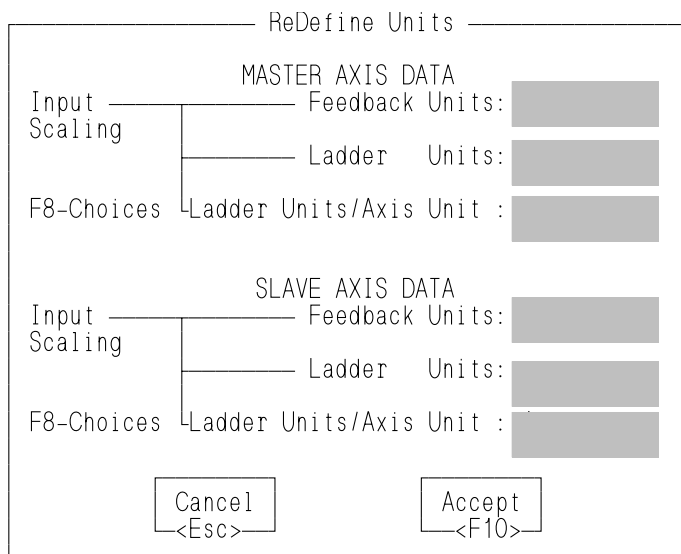
It performs the same save as choosing “Yes” in the Make function box.



If save As is chosen, the Save profile as box appears and you can type in a new filename.

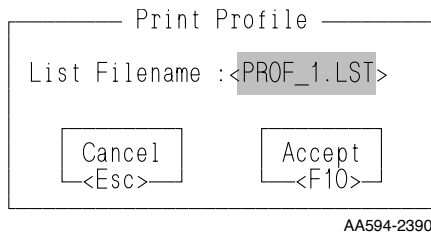
PROF_1.PRO is the example name and can be typed over to change. It is not necessary to type the file extension .PRO. That will be entered automatically.

This command is a copy. The profile with the old name remains in your directory along with the one with the new name.



Redefine units allows you to enter new units to use in the profile.

Press <F10> to accept the new units.



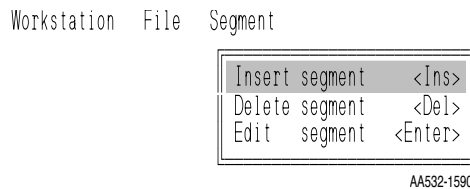
The print command prints the current profile to a text file (.LST file) on disk. You can then send this .LST file to your printer.

You can also go directly to the printer by entering the printer port name in the Print Setup box (i.e., LPT1). No .lst file is created.

The current filename appears in the List Filename. Press <F10> to create the .LST file.

Segment menu

Under the segment menu, you can use the Insert segment command to add a segment anywhere in the profile being defined. Place the selection bar on the segment that occurs *after* the position where you want to insert a new segment, fill in the Insert segment box, and press <F10>.



The Delete segment command allows you to delete any segment in the profile you are defining. Place the selection bar on the segment you want to delete and press .

The Edit segment command lets you alter a segment in the profile you are defining. The Edit segment box appears with the master distance, slave distance, and ending ratio for the segment displayed. Type in new values as needed.

CHAPTER 8 UDFB Guidelines

Introduction

The user defined function block (UDFB) feature will allow you to convert the logic in a module (.LDO) into an individual function block. When you create a function block, it is placed in a library. Whenever PiC(Servo)Pro is executed, the UDFB will be available through this library under the Functions menu. It can then be used in a network of any module you create.

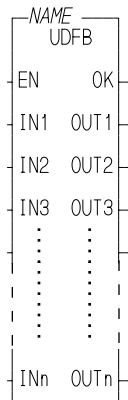
Using the UDFB feature results in an application program that is better organized, more readable, and easier to maintain. Some of the ways the UDFB can be used are listed below.

- UDFBs can segregate the logic for a section of an application program (i.e. diagnostics, fault logging, etc.)
- "Custom" functions with a higher level of functionality can be created (i.e. analog output with ramping).
- A single UDFB can replace in-line programming of repetitive machine functions (i.e. axis 1 fault control, axis 2 fault control, etc.)

The form of a UDFB is shown in Figure 8-1. The EN and OK are required. The remaining inputs and outputs are determined by you when creating the module.

The user defined function block follows the same conventions as the standard function blocks found in PiCPro. It must be declared in the software declarations table when it is used in the network of a module. You assign a name to it when it is declared.

FIGURE 8 - 1. Template for a User Defined Function Block



The first section of this chapter offers guidelines to follow when creating a module that can be converted into a UDFB.

The second section presents an example LDO and the procedure to follow to convert it to a UDFB.

Creating a Module to Convert to a UDFB

- **Naming the module**

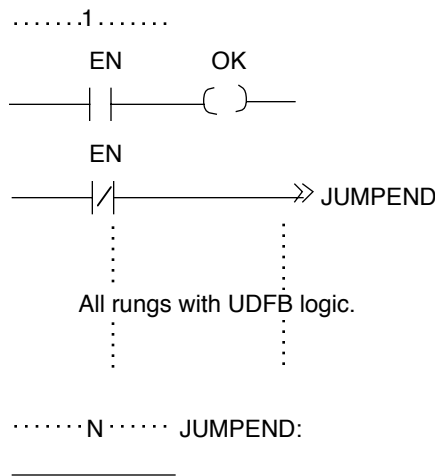
The name of the module will become the name of the UDFB. Be sure the name you choose does not already exist in another module you want to convert to a UDFB or in a standard function or function block supplied by PiC(Servo)Pro.

- **Contents of the module**

Only the logic you want handled by the UDFB should be included in the module. The size of any UDFB can be up to 64K.

- **Programming Note**

The logic contained in the UDFB is scanned regardless of whether or not there is power flow into the EN input. Therefore, in the majority of cases, you will want to include the following lines of logic in your UDFB module.



AA889-3391

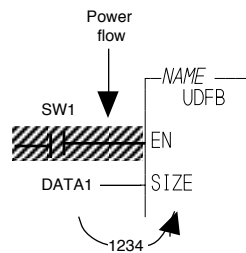
This ensures that you will always have control of the EN and OK in any module you use the function block in. It causes the program to jump to the end of the UDFB logic if the EN is not set. This prevents the logic in the function block from being scanned unless the EN is set.

The only situation where you would not want to control the EN and OK in this way is if you ever create a UDFB which would need to work when the EN is not set. The TOF (timer off) standard function block is an example. The function block continues to execute after the EN is reset.

- **Data handling**

Data is passed to the UDFB from its inputs in one of two ways depending on its type.

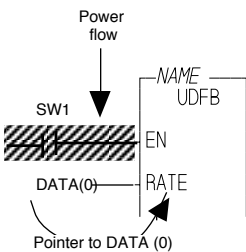
1. If the internal input is any variable other than a structure, string or an array, then the *value* of the external input is passed to the UDFB. This value is acted upon internally in the UDFB leaving the external variable unchanged.



The value of DATA1 (1234) is copied into the variable SIZE when power flow at EN occurs.

AA896-3691

2. If the internal input is a structure, string, or an array, then a *pointer* is passed to the UDFB. This pointer gives the location of the associated data in the calling program. The data is acted upon outside the UDFB.



The pointer to the storage location DATA (0) is passed to RATE when power flow to EN occurs. All accesses to RATE within the UDFB actually manipulate the storage locations reserved for DATA.

AA897-3691

IMPORTANT

Data is copied and pointers are passed into a UDFB only when there is power flow to the first input (EN). *String, array, or structure pointers within a UDFB must not be accessed until power flow to the EN has occurred.*

Software Declarations

Variables of any type with optional initial values may be declared for use in the module. However, variables cannot be external, retained, global, or discrete I/O.

NOTE: If you need to add discrete I/O in order to test the module, be sure to remove it from the ladder and the software declarations table before creating the UDFB.

The variables which will become the inputs for the UDFB may be any data type except function blocks.

The variables which will become the outputs for the UDFB may be any data type except function blocks, structures, arrays, and strings.

Inputs that are structures, arrays, and strings all pass pointers to the UDFB and, therefore, cannot be used as outputs. It is possible, however, to design a function block in which the result of the function block being executed ends up in a structure, array, or string used as an input.

UDFB Variables - Inputs and Outputs

- **Naming variables**
The first four characters you assign to the UDFB input and output variables will become the labels in the UDFB template.
- **Marking inputs and outputs**
The inputs and outputs for the UDFB will be designated in the software declarations table with an **I** or **O**. There must always be at least one input and one output for a UDFB.
- **Order of inputs and outputs**
The first input and the first output will always become the EN (enable) and the OK of the function block. Their data type is **BOOL**. They should appear before any other UDFB inputs and outputs in the declarations table.

Additional UDFB inputs and outputs should be entered in the declaration table in the order you want them to appear in the function block template. The inputs appear on the left side of the template and the outputs on the right.

- **Number of inputs and outputs**
The maximum number of inputs or outputs is 64. It is recommended that you keep the number of inputs and outputs to the UDFB to a minimum (under 16). More can be declared if necessary, but transferring all the inputs and outputs to and from a function block does use scan time. There is also the constraint of the 255 element matrix to consider. If you have a large number of inputs, you may want to enter them as a structure using just one input.

Programming Notes

Recursion/Nesting

When one UDFB calls another UDFB or itself, recursion or nesting occurs. When PiCPro is checking function dependencies, deep nesting (more than four levels) can cause a stack overflow in the workstation. The following error message will appear if this occurs.

```
===== WARNING =====  
Deep nesting of UDFBs can cause the stack in the control to overflow with  
unpredictable results. Generally UDFBs should not be nested to more than  
four levels.  
===== Press any key to continue checking the next UDFB =====
```

Size of UDFB Libraries

Because PiCPro copies the entire UDFB library whenever you do a **UDFB build** during program development, it is recommended that you keep the size of the UDFB library under 100K. This will improve the time it takes to build the UDFBs.

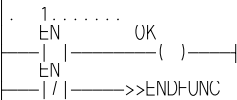
An Example of a UDFB and Module

This section presents an example of a module that will be converted to a UDFB and the procedure for creating and using a UDFB.

The module is called D2A_RAMPLDO and contains five networks as shown below. The function block created from it will provide digital to analog conversion with ramping. The comments in the LDO are shown.

D2A_RAMP LDO Module for UDFB

Workstation Processor Module Declarations Network Element View

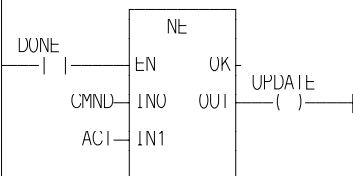
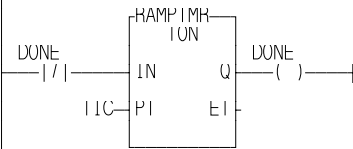


2.....
In addition to EN and OK, this module uses the following variables

CMND = desired command value in counts for the analog output
 RATE = the maximum amount to ramp in counts every time tick
 TIC = the actual analog output is updated every time tick
 MAX = maximum limit for the actual command
 MIN = minimum limit for the actual command
 ACI = actual command for the analog output based on ramp
 This would be the input variable to analog output function.

The timer RAMPIMR will energize DONE for one scan every time tick.

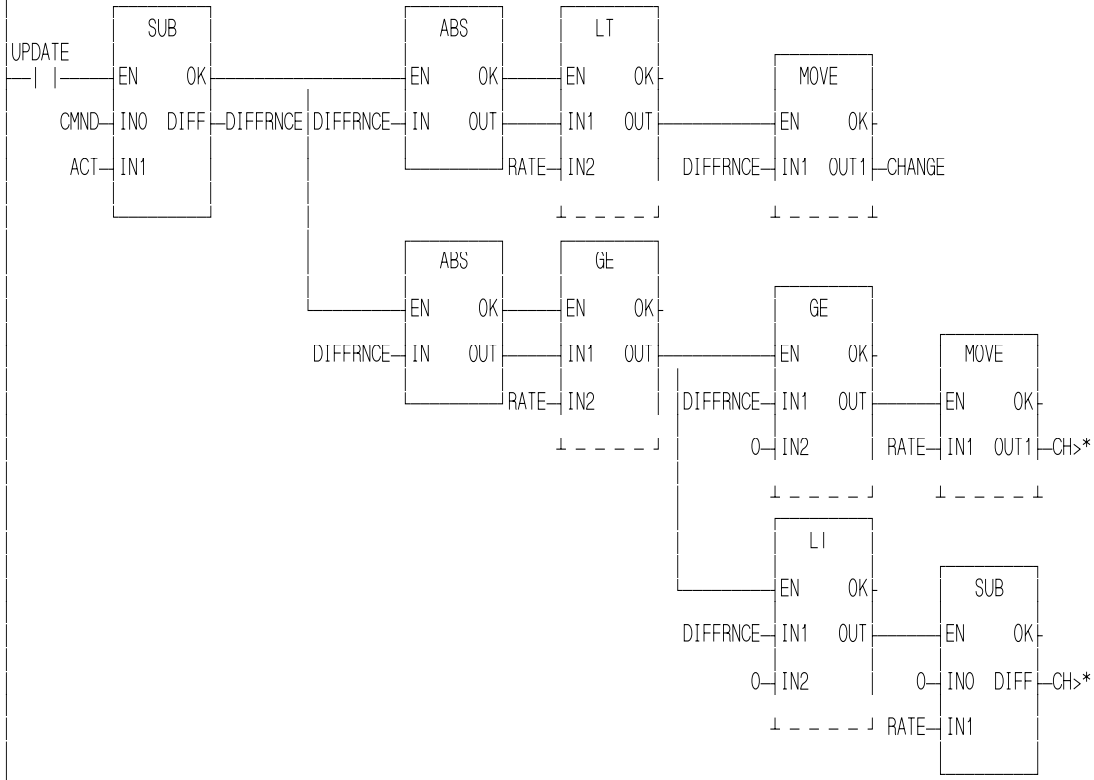
If, when the timer times out, the actual analog output is not equal to the desired command, UPDATE will be set for one scan.



...3.....
 If updating the actual, calculate the difference between command and actual and store in DIFFRNC.

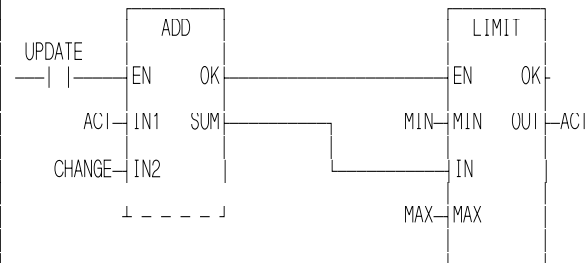
If DIFFRNC absolute value (can be negative) is less than the ramp rate, then the amount to change the command by is equal to DIFFRNC.

If DIFFRNC absolute value is greater than or equal to the ramp rate, then the amount to change the command by is equal to
 RAMPRATE if DIFFRNC is positive
 - RAMPRATE if DIFFRNC is negative



* Same as CHANGE

...4.....
 Add the amount of change to the actual analog output command.



...5..... ENDFUNC:

<End of Module>

D2A_RAMP.LDO Net: 5

You must mark all the inputs and outputs for the function block in the software declarations table. Figure 8-2 shows the declaration table for the example ladder. The first input is the EN and the first output is the OK. The remaining five inputs will appear in order on the left side of the D2A_RAMP function block and the one output will appear on the right below the OK as shown below Figure 8-2.

FIGURE 8 - 2. Software Declarations Table for the D2A_RAMP.LDO

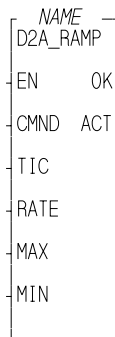
**Inputs and
Outputs for
UDFB**

Inputs and Outputs for UDFB

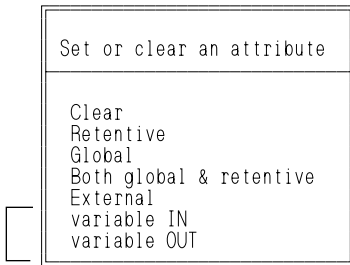
Name	Type	I/O Pt.	Init. Val.	Long Name	Top
EN	I	BOOL			
OK	O	BOOL			
DONE		BOOL			
DIFFRNC		INT			
RAMPTMR		<fb>TON			
CMND	I	INT			
ACT	O	INT			
TIC	I	TIME			
UPDATE		BOOL			
RATE	I	INT			
MAX	I	INT			
MIN	I	INT			
CHANGE		INT			
end_table		void			

Alt-M modifies attribute Press F10 to exit Alt-E enters field Bottom

Example UDFB



Procedure for Marking the Variables



AA1078-4591

Select **Declarations Software** to bring up the software declarations table.

With the cursor in the line with the variable you want to mark as an input or output, press **<Alt M>** to bring up the list of attributes shown on the left.

variable IN - this attribute designates the variable as an input to the user defined function block. When the LDO module is converted to a UDFB, every variable you want to use as an input to the UDFB must have this attribute.

NOTE: The first variable with the "I" attribute in the declarations table will become the EN input of the UDFB. Its data type is BOOL. Additional UDFB inputs should be entered in the order you want them to appear on the left side of the function block.

variable OUT - this attribute designates the variable as an output to the user defined function block. When the LDO module is converted to a UDFB, every variable you want to use as an output from the UDFB must have this attribute.

NOTE: The first variable with the "O" attribute in the declarations table will become the OK output of the UDFB. Its data type is BOOL. Additional UDFB outputs should be entered in the order you want them to appear on the right side of the function block.

Press **I** or **O** to declare that variable as an input or an output for the UDFB.

Press **<F10>** to exit and accept all your entries.

Error Messages

Some of the error messages you may see when working with the UDFB feature include those shown below.

```
At least one input required
Press any key to continue
```

or

```
At least one output required
Press any key to continue
```

If you attempt to create a UDFB without marking the inputs and outputs, the messages shown at the left appear.

```
Name, the first input, is not a BOOLEAN
Press any key to continue
```

AA1079-4591

or

```
Name, the first output, is not a BOOLEAN
Press any key to continue
```

AA1080-4591

If the first input and output (for EN and OK) are not declared as **BOOL** or not marked as **I** or **O**, the messages at the left appear.

```
Invalid symbol attribute in IN1
Press any key to continue
```

If any of the variables are declared as discrete I/O or as external, retained, or global, the message on the left appears.

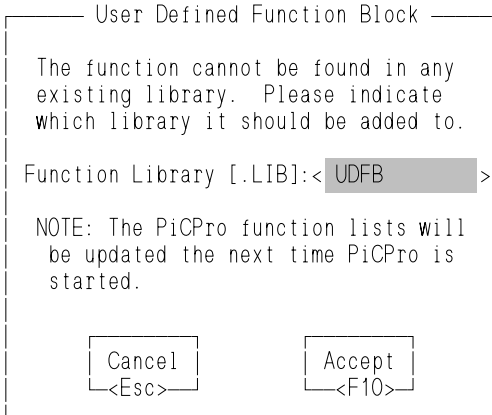
Creating the UDFB

Once the LDO module containing all the logic you want your function block to execute has been written and all the inputs and outputs to the UDFB are marked, you are ready to create the function block by accessing the Module menu.

Workstation Processor **Module** Declarations Network Element View

```
New
Open
Close
Save
save As ...
Download
Monitor
Force
Print
UDFB build
TASK build
Hex-86 dump
Bin-86 dump
build dependency List
find duplicates
```

The UDFB command allows you to convert the logic in the module (.LDO) you have created into a single function block.



MU (Module - UDFB) brings up the dialog box on the left.

Type in the name of the library you want to appear in the **USER** library list of the Functions menu. It will receive the .LIB extension automatically.

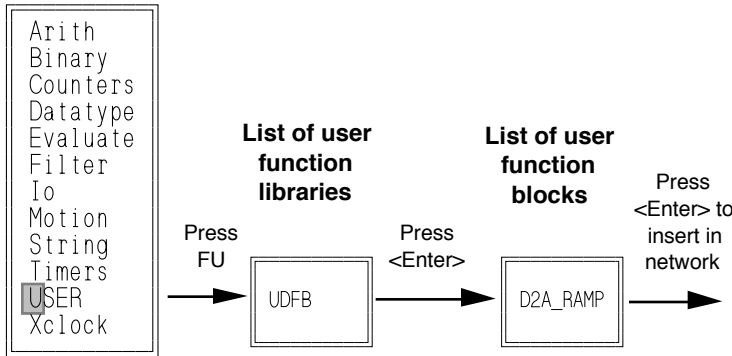
For this example, the library is UDFB.

The new library containing your function block can be accessed from the Functions menu when PiCPro is restarted.

IMPORTANT

Always keep the *UDFB.LIB* file and the corresponding *UDFB.LDO* file(s) in the same directory.

Function Menu



AA1076-4591

When PiCPro is restarted, the Functions menu will contain the word **USER**.

FU (Functions - **USER**) accesses a list of all libraries you have created for UDFBs. Accessing the library will bring up a list of individual function blocks you have created and stored in that library.

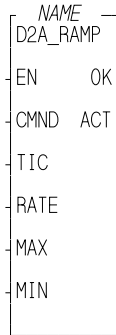
In this example, there is only one example in each list. In the library list, it is UDFB and in the function block list, it is D2A_RAMP.

The UDFB can now be used in any LDO you create. It can also be included as part of an LDO that you want to use to create another UDFB.

Using the UDFB

The function block is now stored in the library and can be inserted in a network in any module you create. The function block created from the D2A_RAMPLDO is shown below with a description of its inputs and outputs.

Digital to analog with ramping



Inputs:

EN (BOOL) - enables execution

CMND (INT) - desired command value in counts for the analog output

TIC (TIME) - designates the time tick in which the actual analog output is updated

RATE (INT) - the maximum amount to ramp in counts every time tick.

MAX (INT) - the maximum limit for the actual analog command

MIN (INT) - the minimum limit for the actual analog command

Outputs:

OK (BOOL) - execution completed without error

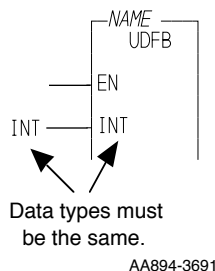
The inputs to the D2A_RAMP function block specify the desired end command (CMND), the ramp update time (TIC), the size of the D/A step (RATE), and the maximum (MAX) and minimum (MIN) output value.

The output to the function block can be fed to any function that requires ramping.

Type Checking

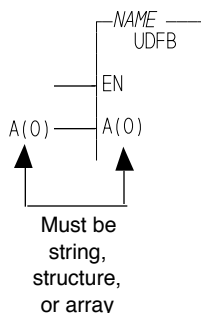
When exiting a network containing a UDFB, type checking of variables will be performed when F10 is pressed. The data types must match.

For example, in the D2A_RAMP function block, the data type at the CMND input was defined as an integer in the D2A_RAMPLDO. When you use the D2A_RAMP function block in a new module, the variable you use at the CMND input must also be an integer.



With variables, if nothing is connected to an input, no data is passed.

With pointers associated with strings, structures, and arrays, type checking is performed to ensure that a memory area is connected. PiCPro will not allow an input requiring a memory area to be left unconnected.



AA895-3691

Editing UDFBs

Depending on the type of change you are making to the UDFB, it can either be done off-line or on-line. PiCPro will determine this and put up one of the two messages shown below when you press MU (Make User). When you make an on-line edit change, one more choice is added to the menu as shown on the right.

Off-Line Editing

```
Function block D2A_RAMP is already defined
in library .\UDFB.LIB
Continue and replace it?

Yes, replace the function in the library
replace the function with a Debug version
No, do nothing
```

On-Line Editing

```
Function block D2A_RAMP is already defined
in library .\UDFB.LIB
Continue and replace it?

Yes, replace the function in the library
replace the function with a Debug version
No, do nothing
no, Patch the function in the running ladder
```

These editing choices are described below.

Yes, replace the function in the library changes the module in the library and requires a full download. If you add any function/blocks and/or declarations that require initialization, you must remake the UDFB and do a full download.

NOTE: Strings always require initialization.

replace the function with a Debug version allows you to create a debug version of the UDFB. Creating a debug version adds additional data bits (40), data bytes (80), and function/jump links (20) to each instantiation of the UDFB for more extensive on-line changes.

NOTE: Minor changes that do not add things like new functions, declarations or jump labels can be made on-line without creating a debug version.

No, do nothing exits this menu without doing anything.

no, Patch the function in the running ladder allows you to make an on-line edit change to this UDFB. This does not change the version of the module in the library, but does allow you to continue to animate and do on-line editing.

The following message appears when the patch download is complete.

```
* Resources Still Available *  
  
40 data bits  
80 data bytes  
260656 code bytes  
98 patches  
20 local label/function links  
26 global label/function links  
  
Use <Ctrl-S> to abort the latest changes  
  
===== Press any key to continue =====
```

Any patches you make are included in the UDFB LDO file only. They will not be included in the UDFB library file until you choose to update the library.

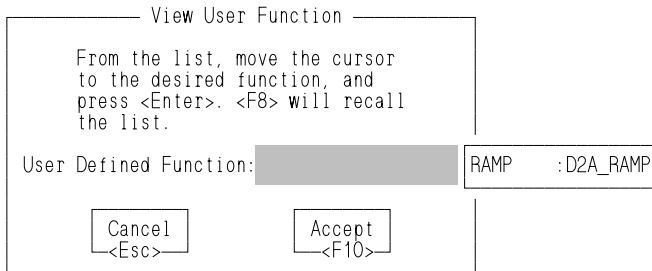
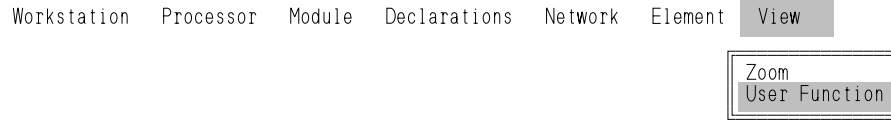
```
The library version of this function does not match  
the .LDO or the version currently in the PiC.  
Do you wish to update the library version?  
  
NOTE: Updating the library will require a full  
download before more on-line-edit of this  
function can occur.  
  
Yes  
No
```

When you attempt to exit PiCPro or open a module, the following message will appear if you have added function/blocks that are not in the current downloaded module. You must update the main module if you want to continue using animation and on-line editing.

```
Because of changes made to one or more UDFBs,  
internal changes are required to the main  
module in order to be able to continue to  
animate and do on-line-edit.  
Do you wish to update the main module?  
  
Yes  
No
```


Viewing the UDFB Module (LDO) from Another Module

When you are running a module containing your UDFB, you may want to view the logic in the module used to create the UDFB. This is possible from the View menu.



AA1077-4591

VU (View - User Function) brings up the View User Function box. A list of UDFBs will appear on the right. (The name you assigned to the function block when it was declared appears to the left in the list box.)

Press <Enter> to choose the UDFB whose module you want to open. Press <F10> to accept your choice. The UDFB module will open. You can run animation to observe the logic within this module for your UDFB.

Use **MO** (Module - Open) to return to your calling program.

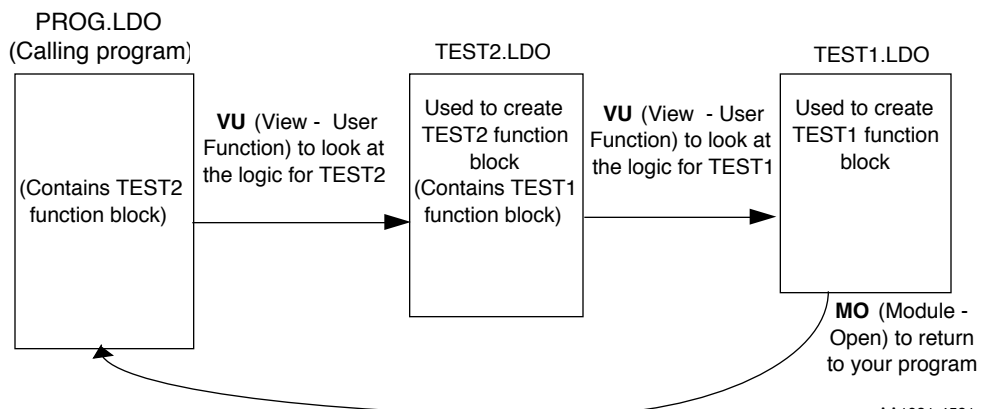
NOTE

The View User Function feature is only available from within a calling program.

An example of how the View User Function feature can be used is shown below.

Module TEST1.LDO is used to create the user defined function block TEST1. The function block TEST1 is used in the logic for module TEST2.LDO to create TEST2 function block. TEST2 is used in your final module which is your calling program.

With the View - User Function feature, you can view and animate each module in descending order. With PROG.LDO running you can view TEST2.LDO. From TEST2.LDO you can view TEST1.LDO as illustrated below.



WARNING

Use discretion when programming tasks in your LDO. You must have a thorough understanding of how they work and what affect they will have on your program to ensure that your program will execute properly.

A TASK is a programming tool that allows you to create a module (.LDO) that will execute at periodic intervals or on the rising or falling edge of a boolean variable from within your main LDO. This provides the ability to schedule events from your main LDO.

Tasks are programmed similar to UDFBs. You convert the TASK LDO into a function block that is stored in a library. Whenever PiC(Servo)Pro is executed, the TASK will be available through this library under the Functions menu. It can then be used in a network of your main LDO.

There are some differences between TASKs and UDFBs as listed below.

- Tasks can access I/O modules located in the main rack of the control.
- A task cannot be programmed within another task or within a UDFB.
- The on-line edit feature cannot be used in a network containing a task.
- Forcing is not allowed in a task.
- When you insert tasks into your main LDO, the function block template is always the same. You cannot program any inputs or outputs for a TASK. They are predefined.

There are three types of TASKs available:

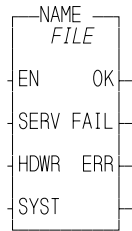
1. **Servo Interrupt Tasks** - triggered on the 1, 2, 4, 8, or 16 ms servo time tick
2. **Hardware Interrupt Tasks** - triggered on a hardware event
3. **System Tick Tasks** - triggered on a multiple of the 10 ms system time tick

NOTE: Variables 44 through 48 are available to be used with servo tasks using the READ_SV and WRITE_SV functions. Refer to the Function/Function Block Reference Guide for information on these variables.

Programming

Every task you create and subsequently use in your main LDO will use the template shown below. FILE will be replaced with the name of your task LDO file when you build the TASK. You will provide a NAME when you declare the function block in the software declarations table of your main LDO.

Template for
Task Func-
tion Blocks



Inputs: EN (BOOL) - enables execution

SERV (TIME) - 1, 2, 4, 8, or 16 ms constant for servo task. Enter in the T# format.

HDWR(BOOL) - I/O point used to trigger the hardware interrupt task must be programmed as **D** (Data **I** (In) or **D** (Data) **V** (in inVerted) which inverts a boolean input. Never program a wire or contact to this BOOL input.

SYST (TIME) - constant or variable based on 10 ms system time tick. If a time is entered that is not a multiple of 10, it will be rounded up to the next 10 ms increment. Time must be less than 10.92 minutes. Enter in the T# format.

Outputs: OK (BOOL) - set if EN is on and the TASK is successfully installed.

Does not indicate whether or not the task has run.

FAIL (BOOL) - set if ERR \neq 0.

ERR (INT) - 0 if no error; \neq 0 if an error.

The EN must be on for a task to run.

The SERV, HDWR, and SYST inputs are mutually exclusive. Only one can be connected at a time depending on whether your task is a servo, hardware, or system interrupt task.

The errors that can appear at the ERR output are listed below.

ERR = 1 Task installation failure

ERR = 2 The time tick requested is larger than 10.92 minutes.

ERR = 3 The servo task is faster than the servo interrupt time.

Creating a TASK

1. Open a new module in PiCPro. The name you give the module will become the name of the TASK function block. Be sure the name you choose does not already exist in a standard or user defined function block or in another task module.
2. Create your TASK program. Keep in mind the following:
 - Tasks can access I/O located in the main rack of the control.
Declare any hardware module whose I/O you use in a task in both the main LDO and in the task LDO hardware declaration tables.
(In the main LDO, all hardware modules must be declared whether the main LDO uses them or not. In the task LDO, only the hardware modules used in the task LDO must be declared in the task LDO.)
 - Hardware outputs used in a task may not be used in the main LDO. Hardware inputs may be used in a task and in the main LDO. Hardware inputs and outputs used in one task may be used in other task(s). Declare them only in the software declaration table of the LDO(s) they will be used in.
 - Inputs are strobed at the beginning of a task. Outputs are strobed at the end of a task. NOTE: When the task is running, only the inputs and outputs in the task, not the inputs and outputs in the main LDO, will be strobed.
 - Do not use timer functions in a task.
 - If software data information in a task needs to be shared with other tasks, declare it as External (E) in the software declarations table of the task LDO. This data must also be declared in the software declarations table of the main LDO whether the main LDO uses it or not. In the main LDO, it is *never* marked as External (E) or Global (G).

NOTE: Never assign the External attribute to direct I/O used in the LDO.

 - To ensure that the data to be transferred between LDOs is from the same scan, interlock multiple externals, arrays, structures and variables. An example of setting semaphore flags as interlocks can be found at the end of this chapter.

The tables that follow summarize some of the rules for making hardware and software declarations in the main and task LDOs.

Description	Hardware Declarations	
	in Main LDO	in Task LDO
Hardware module used in task LDO	Yes	Yes
Hardware module used in main LDO	Yes	No
Hardware module used in both LDOs	Yes	Yes

Description	Software Declarations	
	in Main LDO	in Task LDO
Hardware input used in task LDO	No	Yes
Hardware input used in main LDO	Yes	No
Hardware input used in both LDOs	Yes	Yes
Hardware output used in task LDO	No	Yes
Hardware output used in main LDO	Yes	No
Hardware output used in both LDOs	Not Allowed	
Variable used in task LDO	No	Yes
Variable used in main LDO	Yes	No
Variable used in both LDOs	Yes	Yes Mark External*

*Never mark External any direct I/O used in the LDO.

Hardware modules and I/O points may be used in multiple task LDOs. They must be declared in all task LDOs they are used in.

- When using functions from the I/O or Motion library, the following rules apply.

I/O Functions

Functions in these groups that access an I/O module must be called in the main LDO or within the same task.

ANLGIN ANLGOUT JK THERM
 READFDBK RTDTEMP STEPPER

I/O functions in these groups can be called in the main LDO only.

COMM NETWORK

These I/O functions can be called in both the main LDO and the task LDO.

BAT_OK? PID

Motion Functions

These motion functions can only be called in the main LDO.

STRTSERV	CAM_OUT	MEASURE	REGIST	SCURVE
SC_START	SCA_SEND	SCA_RECV	SCA_CLOS	SCA_ACKR
SCA_REF	SCA_ERST	SCS_SEND	SCS_RECV	SCS_ACKR
SCS_REF				

These motion functions may not interrupt each other, i.e. if one function is accessing a queue, a second function is not allowed to interrupt the first function and access the same queue. If this occurs, the function in the interrupting task will not execute and the OK will not be set.

DISTANCE	POSITION	VEL_END	VEL_STRT	GR_END
RATIOCAM	RATIOPRO	RATIOSLP	RATIOSYN	RATIO_GR
RATIO_RL	REP_END	SYN_END	FAST_REF	LAD_REF
IN_POS?	NEWRATIO	NEW_RATE	ABRTALL	ABRTMOVE
Q_AVAIL?	Q_NUMBER	FAST_QUE	C_RESET	E_RESET
PART_CLR	PART_REF			

These motion functions can be called in both the main LDO and any task LDO.

ACC_DEC	CAPTINIT	CAPTSTAT	CLSLOOP?	CLSLOOP?
COORD2RL	C_ERRORS	C_STOP	C_STOP?	E_ERRORS
E_STOP	E_STOP?	HOLD	HOLD_END	OPENLOOP
P_ERRORS	P_RESET	RATIOSCL	READ_SV	REF_DNE?
REF_END	TME_ERR?	TUNERead	TUNEWrit	WRITE_SV
R_PERCEN	SCR_CONT	SCR_ERR	SCR_PHAS	SCS_CTRL
SCS_READ	SCS_STAT	SCS_WRIT		

The following motion function can only be called once because it does a read and then clear of all but one status flag. After an event occurs, only the first read of the status will recognize the corresponding flag.

STATUSSV

3. After the task LDO is completed, use the **M** (Module) **T** (Task build) command to create the TASK function block.
4. The User Defined Task box appears as shown below. TASK function blocks, just like UDFBs, are stored in a library. Enter the name of the library you want to store the TASK function block in and press <F10> to accept. Tasks and UDFBs may be stored in the same library.

User Defined Task

The function/task cannot be found in any existing library. Please indicate which library it should be added to.

Function/Task Library [.LIB]:< >

NOTE: The PiCPro function/task lists will be updated the next time PiCPro is started.

<Esc> <F10>

The library will appear under the **F** (Functions) **U** (User) command when editing or inserting a network after PiCPro is started again.

IMPORTANT

Always keep the *TASK.LIB* file(s) and the *TASK.LDO* file(s) in the same directory.

5. Once your task(s) has been created, you can call it in your main LDO. When you enter the TASK function in your main LDO, you set up the type of interrupt it will be and when it will run. Some things to keep in mind when using TASKs in your main LDO are listed below.
 - HDWR interrupts can come from the first input in each group of eight on the digital input modules (numbers X.1, X.9, X.17, or X.25) or from the fast input on the encoder/resolver modules. The fast inputs must be numbered as X.1 for channel one, X.3 for channel two, X.5 for channel three, and X.7 for channel four.
NOTE: X represents the slot number for the module.

- Servo interrupt tasks put the heaviest burden on system resources. Always try to conserve system resources by using the appropriate task i.e., do not use a servo task when a system or hardware task can accomplish the same thing.
- Servo interrupts are based on the servo clock. The servo clock can be started in one of two ways:
 1. Calling the STRTSERV function when running servos.
 2. Calling the SERVOCLK function when not running servos.

When tasks are triggered in the main LDO, system tasks will interrupt the main LDO; hardware tasks will interrupt system tasks and the main LDO, and servo tasks will interrupt, hardware tasks, system tasks, and the main LDO. Figures 9-2 and 9-3 illustrate this hierarchy.

If there are multiple tasks of one type in the main LDO, the TASK that is declared first in the software declarations table will execute first.

FIGURE 9 - 1. Priority between the Main LDO, Hardware, and System Tasks

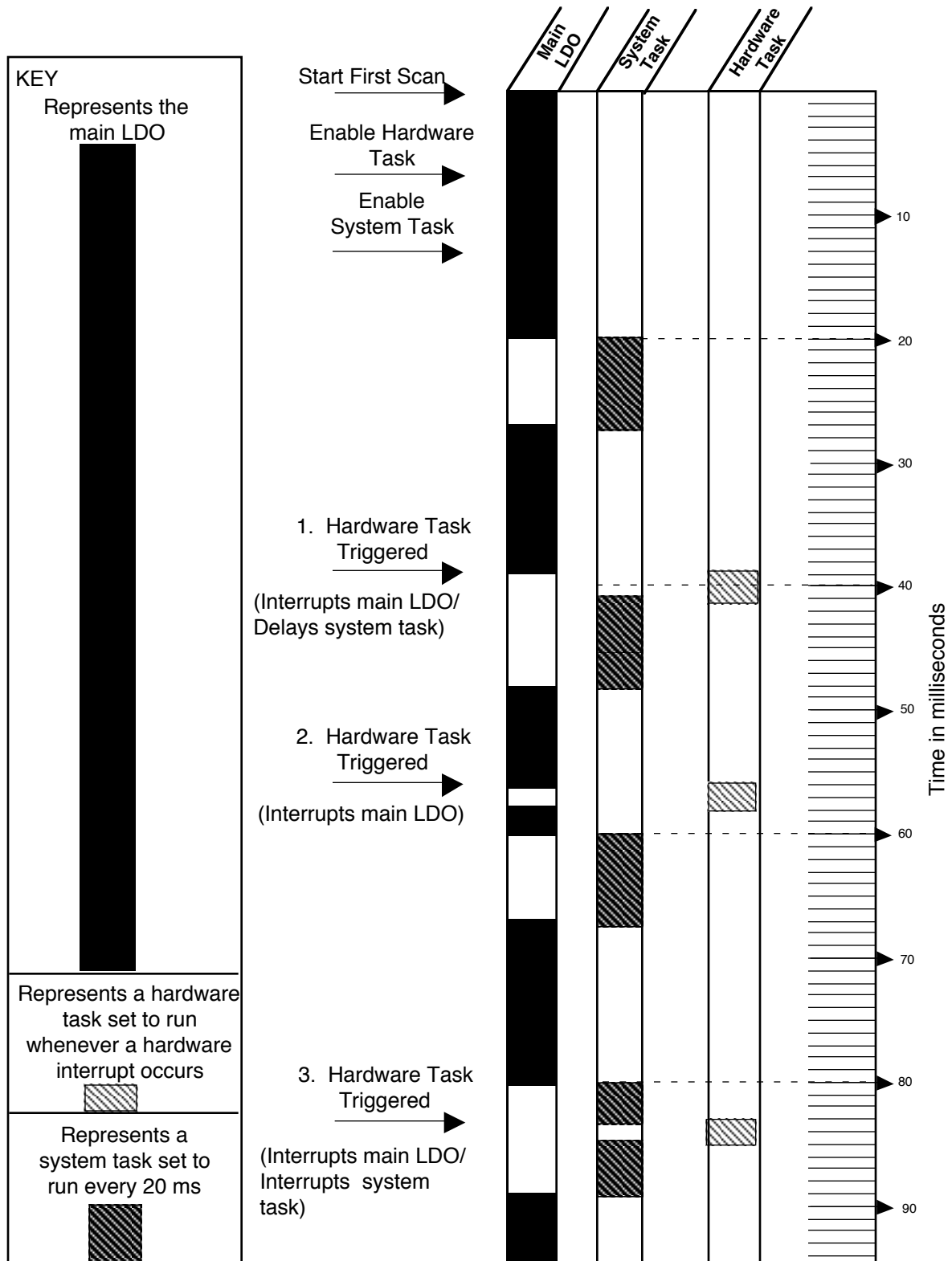
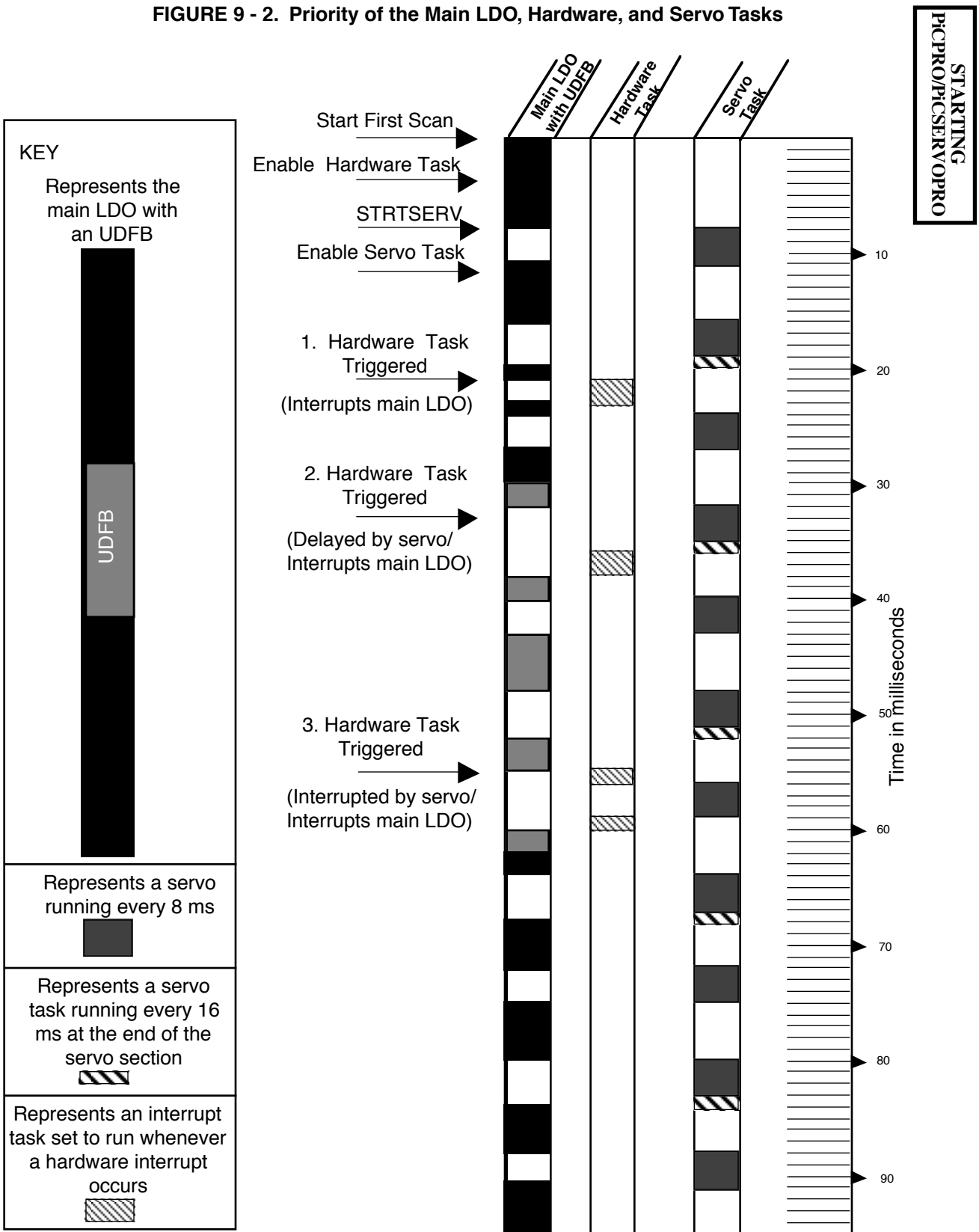


FIGURE 9 - 2. Priority of the Main LDO, Hardware, and Servo Tasks



Interlocking Data when using Tasks

Whenever data is exchanged between the main LDO and a task LDO, there is a possibility of inaccurate data being transferred. This is dependent on when a task interrupts the main LDO or interrupts a lower priority task accessing the same data.

In order to preserve the integrity of the data that is being read/written, you need to program an interlock between the main LDO and the task LDO. One method of interlocking data is to use semaphore flags as explained in the following section.

A semaphore flag refers to a way of communicating between the main LDO and a task LDO when a significant event has occurred.

Setting up Semaphore Flags

In setting up semaphore flags, the main LDO energizes a LOCK flag which prevents the task LDO from reading or writing data while data is being transferred to or from the main LDO. After the data transfer is complete, the main LDO deenergizes the LOCK flag and the task LDO is allowed to read or write the data.

There are two examples presented here. In the first one the data is being transferred from the task LDO to the main LDO and in the second the data is being transferred from the main LDO to the task LDO. In both examples the main LDO has control of the data even though tasks of higher priority are accessing it.

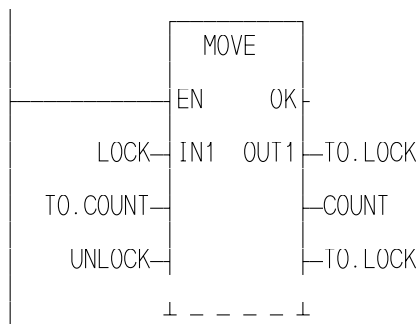
Semaphore Flags when Data Transfer occurs from Task LDO to Main LDO

The MOVE function with a task output (TO) data structure is used in both the main and task LDO. The TO structure acts as a buffer for the data transfer. The TO structure is marked External (only in the task LDO software declarations table, not the main LDO software declarations table) allowing that data to be used by both the main and task LDOs.

In the Main LDO

When the MOVE function in the main LDO is enabled, the following sequence of events occurs.

1. TO.LOCK is energized by moving LOCK into TO.LOCK.
2. TO.COUNT is moved into COUNT.
3. TO.LOCK is de-energized by moving UNLOCK into TO.LOCK.



The variables for the main LDO are defined below.

Variable	Definition
TO	Task output structure
.LOCK	Semaphore flag
.COUNT	Count variable to be written by the task LDO
COUNT	Main LDO count variable
LOCK	IN1 to MOVE function, set to 1
UNLOCK	IN3 to MOVE function, set to 0

The section of the software declaration table for the main LDO is shown below.

Declaration Table in Main LDO

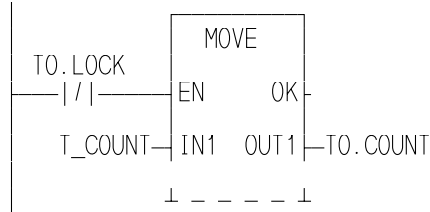
Declaration Table in Main LDO

Name	Type	I/O Pt.	Init. Val.	Long Name	Top
TO	STRUCT				
.LOCK	BOOL				
.COUNT	DINT				
	END_STRUCT				
COUNT	DINT				
LOCK	BOOL		1		
UNLOCK	BOOL		0		

In the Task LDO

In the task LDO, the following sequence of events occurs.

1. If TO.LOCK is off, then move the data. (Step 2)
If TO.LOCK is on, do not move data.
2. T_COUNT is moved into TO.COUNT.



The variables for the task LDO are defined below.

Variable	Definition
TO	Task output structure
.LOCK	Semaphore flag
.COUNT	Count variable to write to the main LDO
T_COUNT	Task variable to be transferred into COUNT in main LDO

The section of the software declaration table for the task LDO is shown below.

Declaration Table in Task LDO

Declaration Table in Task LDO

Name	Type	I/O Pt.	Init. Val.	Long Name	Top
TO	STRUCT				
.LOCK	BOOL				
.COUNT	DINT				
	END_STRUCT				
T_COUNT	DINT				

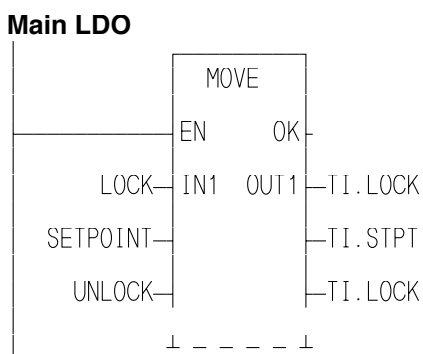
Semaphore Flags for Data Transfer from Main LDO to Task LDO

The MOVE function with a task input (TI) data structure is used in both the main and task LDO. The TI structure acts as a buffer for the data transfer. The TI structure is marked External (only in the task LDO software declarations table, not the main LDO software declarations table) allowing that data to be used by both the main and task LDOs.

In the Main LDO

When the MOVE function in the main LDO is enabled, the following sequence of events occurs.

1. TI.LOCK is energized by moving LOCK into TI.LOCK.
2. SETPOINT is moved into TI.STPT.
3. TI.LOCK is de-energized by moving UNLOCK into TI.LOCK.



The variables for the main LDO are defined below.

Variable	Definition
TI	Task input structure
.LOCK	Semaphore flag
.STPT	Setpoint to be read by task LDO
SETPOINT	Main LDO setpoint variable
LOCK	IN1 to MOVE function, set to 1
UNLOCK	IN3 to MOVE function, set to 0

The section of the software declaration table for the main LDO is shown below.

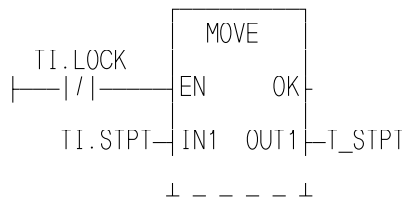
Declaration Table in Main LDO

Name	Type	I/O Pt.	Init. Val.	Long Name	Top
TI	STRUCT				
.LOCK	BOOL				
.STPT	DINT				
	END_STRUCT				
SETPOINT	DINT				
LOCK	BOOL		1		
UNLOCK	BOOL		0		

In the Task LDO

In the task LDO, the following sequence of events occurs.

1. If TI.LOCK is off, then move the data. (Step 2)
If TI.LOCK is on, do not move data.
2. TI.STPT is moved into T_STPT.



The variables for the task LDO are defined below.

Variable	Definition
TI	Task input structure
.LOCK	Semaphore flag
.STPT	Setpoint data to be read by the task LDO
T_STPT	Task variable to hold setpoint from main LDO

The section of the software declaration table for the task LDO is shown below.

Declaration Table in Task LDO

Name	Type	I/O Pt.	Init. Val.	Long Name	Top
TI	STRUCT				
.LOCK	BOOL				
.STPT	DINT				
	END_STRUCT				
T_STPT	DINT				

CHAPTER 10 PiC and SERCOS

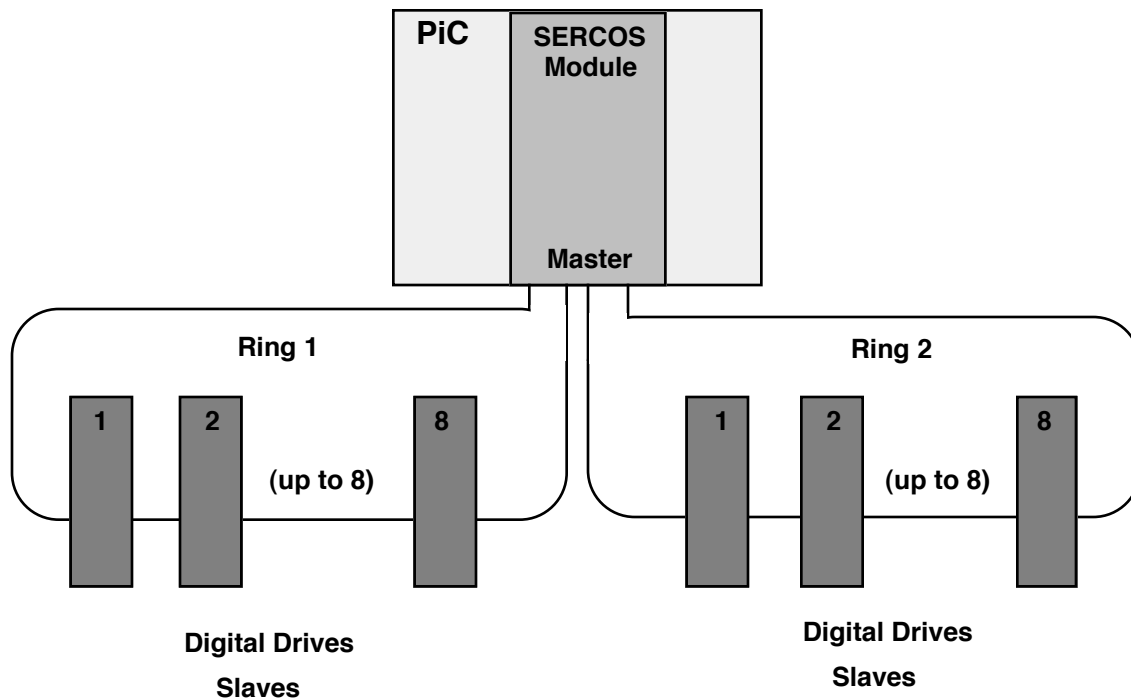
Introduction

SERCOS is a Serial, Realtime Communication System developed to interface with digital controls and drives. Fiber optic cables are used to transmit data serially allowing for improved noise immunity and fast update times. SERCOS allows motion control in velocity, torque, or position modes.

Giddings & Lewis recommends that you acquire a copy of the NEMA SERCOS Specification for reference information, especially if you will be performing advanced control techniques. The SERCOS Specification is available through ANSI. To order call ANSI at 212-642-4900 and ask for Manual IEC 1491.

SERCOS allows you to create a digital instead of an analog motion control network. A typical PiC/SERCOS network is illustrated below. The network consists of up to eight digital drives connected to one master in a ring topology in which messages travel unidirectionally. The network can support two fiber optic rings from one SERCOS module.

Figure 10- 1. PiC/SERCOS Network



NOTE: Currently, the PiC/SERCOS configuration supports a slave controlling a single drive on the ring. It does not support a slave controlling a group of drives as described in the SERCOS Specification.

Background Information

SERCOS Functions

The Motion library contains the following function/function blocks in the groups listed. Refer to the Function/Function Block Reference Guide for a complete description of the function/function blocks.

NOTE
<p>SCA_... function blocks have an AXIS input to identify the SERCOS axis that has been initialized in Servo setup. SCS_... function/function blocks identify the SERCOS axis by slot, ring, and slave numbers.</p>
<p>SCR_... function/function blocks apply to one ring. SC_... function applies to all rings.</p>

DATA Group
SCA_RECV - receives data from the service channel of the AT for a servo axis
SCA_SEND - sends data to the service channel of the MDT for a servo axis
ERRORS Group
SCA_ERST - resets E-errors and closes the loop on a servo SERCOS axis
INIT Group
SCA_CLOS - reads the current position and closes the loop on a servo SERCOS axis
REF Group
SCA_ACKR - acknowledges the reference cycle for a servo axis
SCA_REF - starts the reference cycle on the servo SERCOS axis
SERC_SLV Group
SCS_ACKR - acknowledges the reference cycle for an non-servo axis
SCS_CTRL - writes the control bits to the MDT
SCS_READ - reads cyclic data from the AT
SCS_RECV - receives data from the service channel of the AT for a non-servo axis
SCS_REF - starts the reference cycle on the non-servo SERCOS axis
SCS_SEND - sends data to the service channel of the MDT
SCS_STAT - reports the status from the AT
SCS_WRIT - writes data to the cyclic data of the MDT
SERC_SYS Group
SCR_CONT - continues communication if it was halted after Phase 2
SCR_ERR - reports ring errors
SCR_PHAS - reports the highest phase number completed
SC_START - copies the user data into the SERCOS module

Telegrams

SERCOS has a master/slave communications structure with the PiC as the master and the drives as the slaves. All the communication between the two is done with messages referred to as “telegrams.” A telegram is an ordered bit stream carrying data and timing information.

There are three standard telegrams used in SERCOS communication as described below:

Name Description

1. **MST - Master Synchronization Telegram** - sent by the PiC to the drives to set the timing and synchronization for everything that happens on the network.
2. **AT - Amplifier Telegram** - sent by the drives to the PiC.
3. **MDT - Master Data Telegram** - sent by the PiC to the drives after it receives the last AT in a cycle.

The type of data transferred by these telegrams is determined by the telegram type you enter in SERCOS setup. IDN 15 contains the telegram types available. They are described below.

IDN 15 Telegram Type	Description	IDN 32**
0	No data	(Not Sent)
1	Torque control	1
2	Velocity control, velocity feedback	2
3	Velocity control, position feedback	2
4*	Position control, position feedback	3
5	Position and velocity control, position and velocity feedback	3
6	Velocity control	2
7	User-defined	(Not Sent)

*If the slave will be controlled by the Motion.Lib, 4 is the telegram type number that is entered in SERCOS setup.

**IDN 32 is sent by the SERCOS software if telegram type 1 through 6 is entered in SERCOS setup.

SERCOS Slaves (using SCS_... function/function blocks)

Figures 10.2 and 10.3 indicate how some of the SERCOS function/function blocks are used in the communication structure when working with slaves that have not been initialized in Servo setup.

Figure 10-2. The MDT Structure

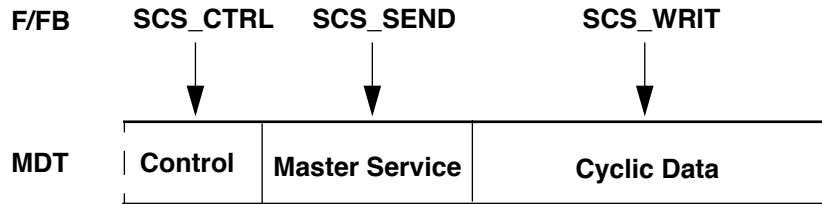
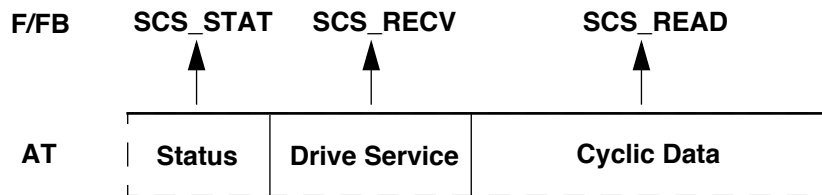


Figure 10-3. The AT Structure



Servo SERCOS Axes (using SCA_... function blocks)

Figures 10.4 and 10.5 illustrates what happens within the communication structure when working with servo axes that have been initialized in Servo setup.

Figure 10-4. The MDT Structure

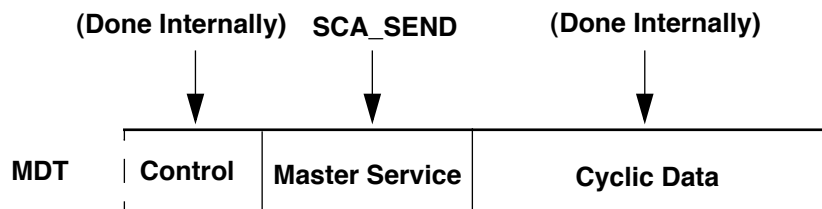
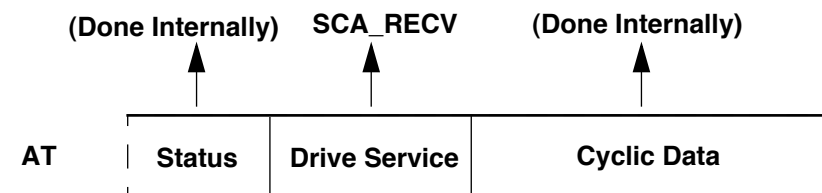


Figure 10-5. The AT Structure



IDNs

SERCOS has a system of identification numbers (IDNs) used to access specific data. You can find a list of these IDNs in the SERCOS specification. There is a standardized set covering the following categories:

- | | |
|----------------------------------|-------------------------------------|
| 1. General parameters | 6. Definitions of telegram contents |
| 2. Diagnostics | 7. Drive operation modes |
| 3. Lists of operation data | 8. Standard operation data |
| 4. Internal and external signals | 9. Drive parameters |
| 5. Manufacture specifications | |

There is also a user-defined set of IDNs available.

Below is a list of IDNs that must be supported by any slave device you are using. They are listed in the order they are worked through as Phase 2 and 3 proceed. For complete descriptions of these IDNs, consult the SERCOS specification.

IDN	Description
00003	Shortest AT Transmission Starting Time (T1min)
00004	Transmit/Receive Transmission Time (TATMT)
00005	Minimum Feedback Processing Time (T4min)
00088	Receive to Receive Recovery Time (TMTSY)
00090	Command Value Proceeding Time (TMTSG)
00096	Slave Arrangement 0
WRITE	Description
00001	Control Unit Cycle Time, TNcyc
00002	Communication Cycle Time, TScyc
00006	AT Transmission Starting Time (T1)
00007	Feedback Acquisition Capture Point
00008	Command Value Valid Time
00009	Position of Data Record in MDT
00010	Length of Master Data Telegram
00015	Telegram Type Parameter
00032	Primary Operation Mode
00089	MDT Transmission Starting Time
R/W	Description
00099	Reset Class I Diagnostic
00127	CP3 Transition Check
00128	CP4 Transition Check

Phase 2

Phase 3

Note on Working with Procedure Command Function IDNs

There are two types of IDNs used in SERCOS.

1. **Parameter data IDNs** - These handle simple commands (i.e., IDN 100 where the value for gain can be read or written). You define the values for these IDNs.
2. **Procedure command function (PCF) IDNs** - These are IDN commands that take a longer time to execute (i.e., IDN 262, load parameter data or homing cycle). With PCFs the one word DATA values hold bit patterns that relay the progress of the function. The MDT has two bits and the AT has five bits. All PCFs use these bits the same way. This bit field provides the communication between the master and the slave.

Follow these steps to use a PCF IDN.

1. Start the procedure by sending the PCF IDN with the data value of 3 using SCS_SEND or SCA_SEND. This sets (bit 0 = 1) and enables (bit 1 = 1) the PCF. SIZE for PCF IDNs is one word.
2. In the SCS_RECV or SCA_RECV function blocks, enter a 1 as the ELEM member of the DATA structure.
3. Continually read the PCF IDN value using SCS_RECV or SCA_RECV function blocks. If the command was received by the drive, a status of 7 will be read. This means the drive acknowledges the command set (bit 0 = 1) and enabled (bit 1 = 1). If no error occurs and the procedure is complete, a status of 3 is returned. This means the drive acknowledges the command set (bit 0 = 1), enabled (bit 1 = 1), and executed (bit 2 = 0). This may take only a few SERCOS updates or several minutes depending on the PCF. If an error occurred during the procedure, bit 3 will equal 1.
NOTE: There is a change bit (bit 5) in the status word which can be used to avoid continually reading the PCF IDN value as described above. See the SERCOS specification for more information on this change bit.
4. After the read value of the PCF IDN equals status of 3, the command set (bit 0 = 1), the enabled (bit 1 = 1), and executed (bit 2 = 0), or the read value has status bit 3 set (≥ 8), the PCF must be sent again using SCS_SEND or SCA_SEND with data = 0. This cancels the procedure by sending the PCF IDN with bit 0 = 0.

This PCF procedure and other advanced SERCOS features are described in the SERCOS specification. Refer to section 7.4.4 of the NEMA SERCOS specification for more detailed information on PCFs.

NOTE: PCFs may not be used in the SERCOS setup list.

Phases

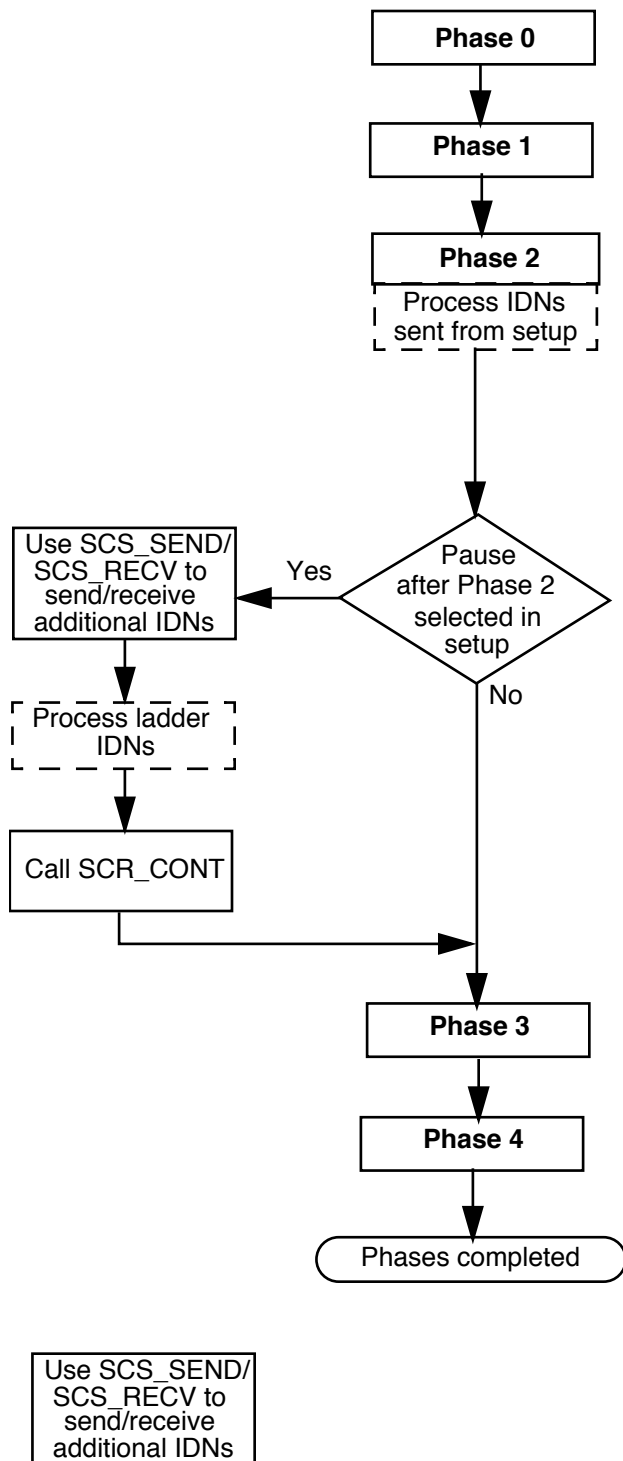
During initialization, the SERCOS system moves through five communication phases (0 through 4) before normal communication can begin. These are summarized below.

Phase # Description

- Phase 0** Closes the communication ring and allows initialization to proceed.
- Phase 1** An identification phase where the PiC addresses each drive individually and waits for an acknowledgment.
- Phase 2** Sets parameters for cyclic data transmission.
- Phase 3** Sets parameters for non-cyclic or service channel data transmission.
- Phase 4** The SERCOS ring can begin normal operation as defined by the application.

The movement through these five communication phases occurs in sequential order. In SERCOS setup, it is possible to decide to halt the movement after the completion of Phase 2 allowing you to send additional IDNs specific to your application which you did not enter in SERCOS setup and then resume moving through Phase 4. At the end of Phase 4, you have another opportunity to send IDNs as illustrated in the flow chart.

Figure 10-6. IDNs and Phases



NOTE: SCA_SEND and SCA_RECV function blocks used with servo SERCOS axes cannot be used until the servos have been initialized (START_SV function called successfully). This can only occur after Phase 4.

NOTE: It is not possible to go back to a previous phase unless you go all the way back to Phase 0 and reinitialize.

Cyclic Operation

After initialization is complete, SERCOS transmits critical data such as command values, feedback values, etc. in synchronized, cyclic form. This data is updated cyclically in real time at the update rate set in the PiC.

Service Channel

Non-critical data such as diagnostics, loop gains, etc. are transferred over a service channel. This data is transferred once each cycle and goes both ways. It may take several cycles to transfer data. The transfer is initiated by the PiC. Figures 10.2 to 10.5 illustrate this transfer.

Notes on using Standard Motion.Lib Functions and Variables with SERCOS

This section summarizes things you should be aware of when using SERCOS feedback and the motion library of PiCServoPro.

READ_SV/WRITE_SV Functions

These READ_SV and WRITE_SV variables cannot be used when using SERCOS feedback on an axis.

Var #	Name
9	Fast input position (Hardware)
10	Registration/referencing position change
11	Consecutive bad marks
19	Fast input direction
20	Fast input distance
24	Registration switch
27	Backlash compensation
28	TTL feedback
29	Reference switch position
46	Set user PID command
47	User PID command
48	Disable servo software

All other READ_SV and WRITE_SV variables can be used with an axis using SERCOS feedback.

TUNERead/TUNEWRIT Functions

The filter variable is the only one that can be read and written by these functions when using a SERCOS axis. The remaining TUNERead/TUNEWRIT variables cannot be used with a SERCOS axis.

CLOSLOOP/OPENLOOP Functions

The CLOSLOOP function is not called on a SERCOS axis. The SCA_CLOS function block is used to close the loop on the SERCOS axis. The top three bits of the SERCOS control word are set on subsequent SERCOS updates. When the OPENLOOP function is called, the top three bits of the SERCOS control word are cleared on subsequent SERCOS updates.

E_RESET Function

The E_RESET function is not called on a SERCOS axis. The SCA_ERST is used to reset E-stop conditions on a SERCOS axis. The SCA_ERST function will read the current position of the drive prior to closing the position loop.

REGIST Function

The registration function cannot be used on a SERCOS axis.

Reference-Related Functions

The SCA_REF (servo) and the SCS_REF (non-servo) function blocks are used to perform referencing functions with a SERCOS axis.

The reference-related functions in PiCServoPro on the left below cannot be used with a SERCOS axis. The functions on the right can be used with a SERCOS axis.

Not Available with SERCOS	Available with SERCOS
FAST_REF	PART_REF
LAD_REF	PART_CLR
REF_DNE?	
REF_END	

STRTSERV Function

The SERCOS ring must be into Phase 4 of its communication cycle before the servos can be initialized. Call STRTSERV after Phase 4 has started.

An additional error can appear at the ERR output with a SERCOS axis.

ERR#	Description
5	The axis update rate in PiCServoPro is different from the SERCOS ring update rate.

10.1 Startup Procedure

This section will explain what you need to do to start up your system.

SERCOS Setup

You define the SERCOS configuration for all the SERCOS axes using the SERCOS setup tool in PiCServoPro.

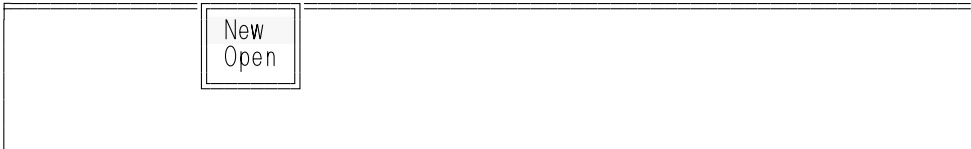
1. Run PiCServoPro by typing **picservo** at the DOS prompt.
2. When this screen is displayed, type **PE** (Processor, sErcos setup) to start the SERCOS setup program.

```
Workstation Processor pProjects
```

```
Control programmer (PICPRO)
Servo setup and tuning (SRVSETUP)
sErcos setup (SRCSETUP)
Axis profile setup (PICPFL)
```

3. Type **FN** (File, New) to create a new SERCOS setup file [or **FO** (File, Open) to open an existing SERCOS setup file].

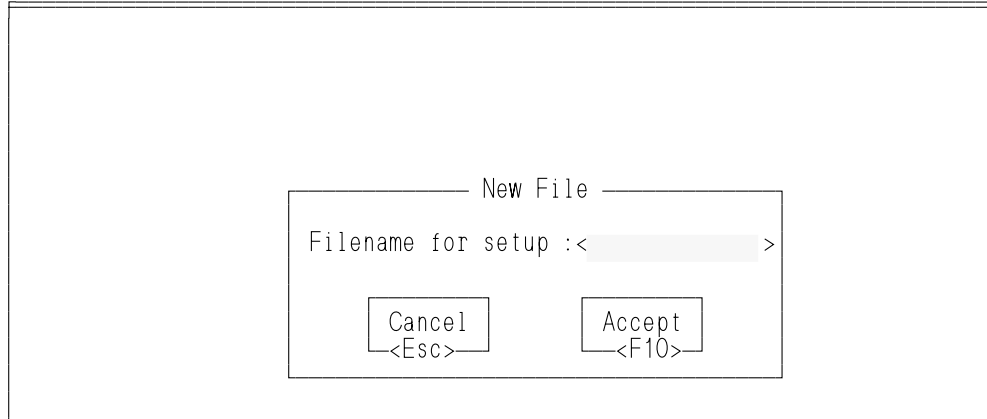
```
Workstation File
```



The screenshot shows a window titled 'Workstation File'. A menu box is open, displaying two options: 'New' and 'Open'. The 'New' option is highlighted with a grey background.

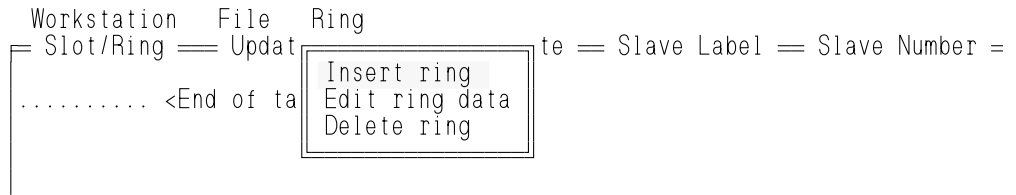
4. When the New File box appears, type in the name of the file and press <F10> to accept or <Esc> to cancel.

```
Workstation File
```

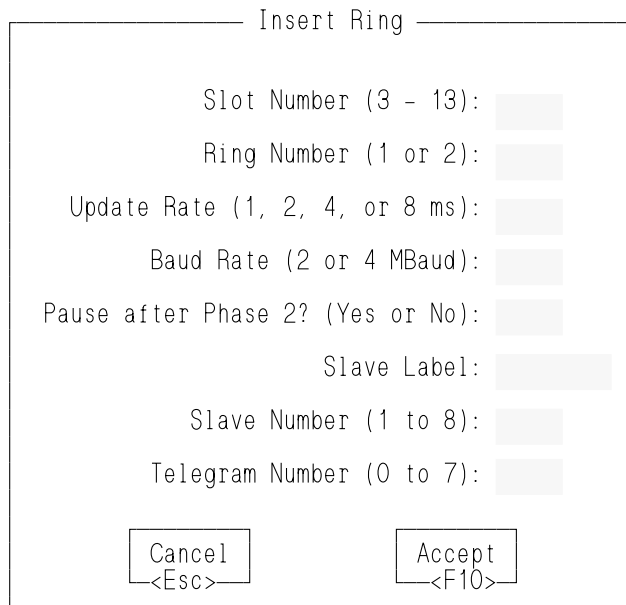


The screenshot shows a window titled 'Workstation File'. A dialog box titled 'New File' is open. It contains a text input field labeled 'Filename for setup :<' followed by a cursor and a '>' character. Below the input field are two buttons: 'Cancel' with '<Esc>' below it, and 'Accept' with '<F10>' below it.

- The following appears. Press **RI** (Ring, Insert) to insert one of two rings available with each SERCOS module.



The following dialog box appears. Fill in the information according to the requirements of your application to enter the first ring and the first slave axis.



Slot Number

The Slot number identifies which slot the SERCOS module is installed in the PiC rack.

Ring Number

The Ring Number identifies whether this slave axis is on the first or second ring on a SERCOS module.

Update Rate

The update rate for any ring cannot exceed the PiC update rate. The PiC update rate is the fastest of all servos initialized. If the SERCOS slave is a servo axis, the ring update rate must be the same as the axis update rate.

Background Information

Baud Rate

Currently, most devices operate at 2MBaud.

Pause after Phase 2?

A Yes allows you to send/receive additional IDNs if required.

Slave Label

You can assign a name to the slave axis here.

Slave Number

The slaves on a ring must be numbered sequentially. For example, if you have three slave axes on a ring, number them 1, 2, 3; not 1, 2, 4. The number entered here must match the address switch set on the slave.

NOTE: The Slot Number, Ring Number, and Slave Number correspond to the SRS input on the SERCOS functions which identify this slave axis.

Telegram Number

Enter the type of telegram data you will use in your application. If the slave will be controlled by the Motion.Lib, 4 is the telegram type number that must be entered in here.

IDN 15 Telegram Type	Description
0	No data
1	Torque control
2	Velocity control, velocity feedback
3	Velocity control, position feedback
4	Position control, position feedback
5	Position and velocity control, position and velocity feedback
6	Velocity control
7	User-defined

Press <F10> to accept (or <Esc> to cancel).

6. To enter additional slave axes for the ring, press **SI** (Slave, Insert).

Workstation	File	Ring	Slave		Label	Slv #	Tel #
Slot/Ring	Update						
			Insert slave Edit slave data Delete slave eNter IDN numbers				

7. The following dialog box appears. Fill in the information.

```

      Insert Slave
    -----
      Slave Label:
    Slave Number (1 to 8):
    Telegram Number (1 to 7):

    [Cancel]      [Accept]
    <Esc>         <F10>
  
```

8. (Optional) To enter the number of any additional IDNs you may want to send, press **SN** (Slave, eNter IDN number). The following screen appears.

```

IDN Number
Slave Label: AXIS1      Slave Number: 1      Slot/Ring: 3/1
----- IDN Number ----- Size in Words ----- Data -----
..... <End of IDN list> .....
  
```

9. Press **II** (IDN Number, Insert IDN number).

```

IDN Number
Slave Number: 6      Slot/Ring: 3/1
Edit IDN Number  Size in Words ----- Data -----
Delete IDN Number IDN list> .....
Insert IDN Number
  
```

Background Information

10. The following screen appears. Enter the information required. Press <F10> to accept the data and return to the previous menu.

```
----- Insert IDN -----  
  
IDN Number:   
Size in Words (1 or 2):   
Data:   
  
   
<Esc> <F10>
```

11. After all the data has been entered, you will make a function that will be used in your LDO. Press **FM** (File, Make function).

```
Workstation File Ring Slave  
  
  
  
  
  
  
  

```

12. The box shown below appears. Enter the name of the function. Enter the name of the library the function should be stored in. A Yes in Save Setup Data means that the SERCOS setup data will automatically be stored to disk. A No means you will have to use either the Save or save As option under the File menu to save it.

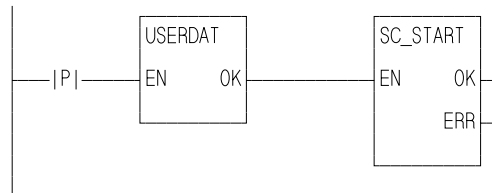
```
----- Make function -----  
  
Function name :   
Library Filename : <  >  
Save Setup data:   
  
   
<Esc> <F10>
```


Application Notes for Your Ladder Program

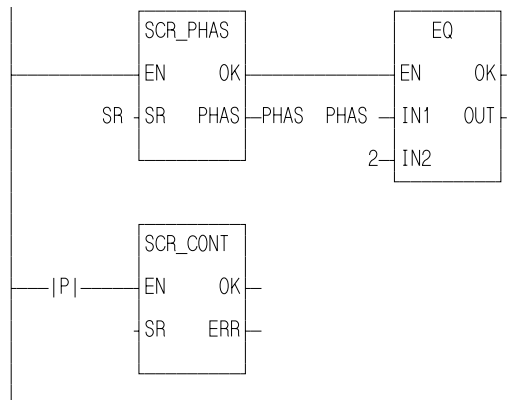
This section contains programming information.

SERCOS Startup

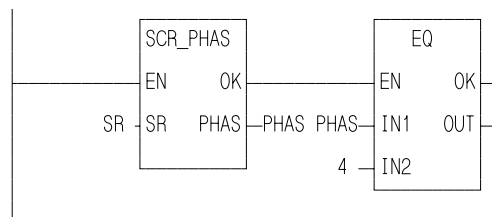
The SERCOS setup data is made into the USERDAT function in SERCOS setup. The SC_START function installs all the data to the SERCOS module specified in the setup data.



OPTIONAL: If you will be pausing after phase 2 to send additional IDNs using the SCS_SEND function, call the SCR_PHAS until it indicates that phase 2 is completed. After all the IDNs have been sent, use the SCR_CONT function to continue through phase 4.



Call the SCR_PHAS function until phase 4 is completed. After phase 4 is completed, the SERCOS functions can be used to send or receive information. The SERCOS or Motion functions can be used to control position.

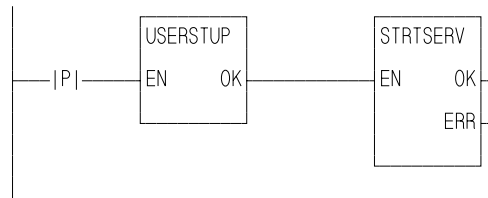


Background Information

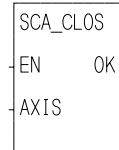
You can control axis position using either servo or non-servo SERCOS function/function blocks. In SERCOS setup, the telegram type must be Telegram 4 position mode. As defined by the SERCOS specification, with Telegram 4 IDN 47 is cyclic data sent from the PiC to the slave. IDN 51 is cyclic data sent from the slave to the PiC. When using a servo axis, SERCOS is selected as the feedback type in Servo Setup.

Servo SERCOS Axis Position Control

The servo setup data is made into the USERSTUP function in servo setup. The STRTSERV function installs all the data.

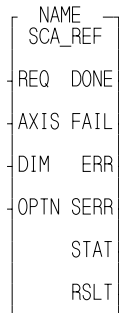


To close a SERCOS axis loop, use the SCA_CLOS function. It will read IDN 47, update the servo data with the new position, send the value as commanded position, and set the control bits to cause the drive to close it.

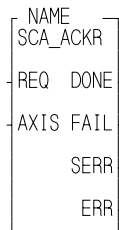


Referencing with the SCA_REF Function Block

The SCA_REF function block performs all the functions of SCS_REF and also updates the servo data with the new position.



Use the SCA_ACKR function block to acknowledge that the reference cycle is complete. Control will return to the PiC after this function block is called.



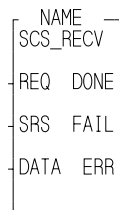
Background Information

Non-servo SERCOS Axis Position Control

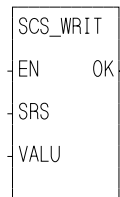
The SCS_WRIT and SCS_READ functions are used to write and read cyclic data.



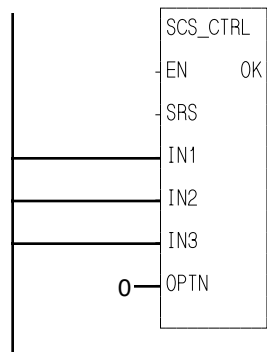
When operating in the position mode, IDN 47 contains the position command value. You must read this value from the drive before the loop is closed using the SCS_RECV function block.



Write this position value using SCS_WRIT before the control bits are set using the SCS_CTRL function.

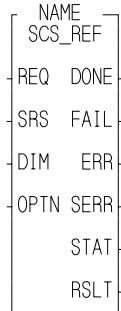


When the control bits are set, the drive is told to go to the commanded position using IDN 47. Since this is where the drive is, there will be no movement.

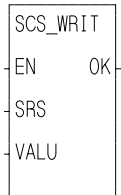


Referencing with the SCS_REF Function Block

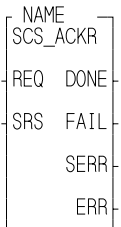
The current position from the drive must be read to perform referencing. The reference cycle within the drive defines the value for IDN 47. The SCS_REF function reads the value for IDN 47 as the last step in the reference process and returns this value in the RSLT output.



The RSLT value must be sent by the SCS_WRIT function before calling the SCS_ACKR function.



Again, the drive is told to go to the commanded position in IDN 47.



Rollover on Position

PiC900 with PiCServoPro and Motion.Lib

To use the rollover on position feature with a servo axis, a rollover value is either entered in servo setup or written with variable 12.

SERCOS Drive

To use the rollover on position feature (referred to as modulo) with a SERCOS drive, IDN 76 (bit 7) is used to turn the feature on and IDN 103 defines the rollover value.

Application Note

When using the rollover on position feature with a servo SERCOS axis, you must combine both methods described above.

To implement rollover on position on a servo SERCOS axis the following must be true:

IDN 76 Bit 7 = 1 **IDN 103 = the rollover value** **In Servo Setup or Variable 12 = rollover value**

To prevent rollover on position on a servo SERCOS axis the following must be true:

IDN 76 Bit 7 = 0 **IDN 103 = NA** **In Servo Setup or Variable 12 = 0**

CHAPTER 11 Stepper Axis Module

Introduction

Stepper motors can be controlled by:

- The stepper motor control module (SMCM) using the PiCPro stepper functions

or

- The stepper axis module (SAM) using Servo Setup and the move types available in the PiCPro motion library. It can be a master or a slave in the application.

This chapter covers the stepper axis module. Any move type from the motion library can be used to perform motion control with the stepper except those move types requiring a fast input. There is no feedback from the stepper axis module.

Servo setup is used to set up the stepper axis module and create a start servo function.

Servo Setup

Define Axis

Axis Label :

Feedback Module :

Input slot :

channel :

Output slot :

channel :

Feedback Type

- Encoder
- Resolver
- Analog Input
- TTL Input
- SERCOS
- Stepper

Cancel <Esc> Accept <F10>

The Feedback Module field allows you to enter the feedback device used for the axis. Press <F8> to bring up a list of types. Move the cursor to the feedback device you are using and press <Enter>.

Fill in the appropriate slot/channel information and press <F10>.

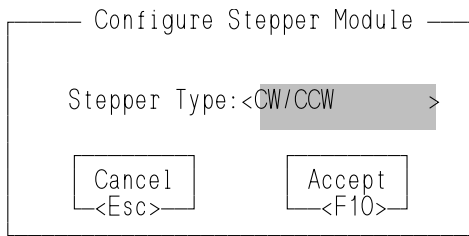
NOTE: When SERCOS is the feedback type, enter the ring number in the channel entry.

NOTE: When Stepper is the Feedback type, the input slot/channel entry must match the output slot/channel entry.

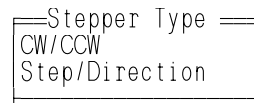
Press <Esc> to cancel data

Notes on using Motion Library Functions and Variables with the Stepper

Stepper Axis Setup



Depending on the type of stepper you are using, select from the <F8> list.



The stepper type defaults to CW/CCW.

Once all the setup data has been entered, define the servo function. This function will be stored in the servo library and can then be called into your ladder program to initialize the setup data for your application.

Notes on using Motion Library Functions and Variables with the Stepper

This section summarizes things you should be aware of when using the stepper and the motion library of PiCServoPro.

READ_SV/WRITE_SV Functions

These READ_SV and WRITE_SV variables cannot be used when using the stepper on an axis.

Var #	Name
4	Position error
9	Fast input position (Hardware)
10	Registration/referencing position change
11	Consecutive bad marks
19	Fast input direction
20	Fast input distance
24	Registration switch
27	Backlash compensation
28	TTL feedback
29	Reference switch position
46	Set user PID command
47	User PID command
48	Disable servo software

All other READ_SV and WRITE_SV variables can be used with a stepper axis.

NOTE: Feedback units are stepper units. Ladder units may still be used.

TUNERead/TUNEWRIT Functions

The filter variable is the only one that can be read and written by these functions when using a stepper axis. The remaining TUNERead/TUNEWRIT variables cannot be used with a stepper axis.

CLOSLOOP, OPENLOOP, REGIST, and MEASURE Functions

These functions cannot be used on a stepper axis.

Reference-Related Functions

The reference-related functions in PiCServoPro on the left below cannot be used with a stepper axis. The functions on the right can be used with a stepper axis.

Not Available with Stepper	Available with Stepper
FAST_REF	PART_REF
LAD_REF	PART_CLR
REF_DNE?	
REF_END	

STRTSERV Function

Call STRTSERV to initialize the stepper axis and begin the stepper motion.

ERRORS

There is no loss of feedback or excess error. If an E-stop error occurs, the command to the stepper will be zeroed.

NOTES

Appendices

Appendix A - Module Filenames

The following table shows the files that are created during the programming process.

Extensions	Type of file	Created/updated upon
Filename is module name		
LDO Ladder Diagram Object	ladder or network logic file	Save
LBK Ladder BacKup	backup copy of the LDO file - is the version previous to the version created with last Save	Save
REM REMArks	the documentation or comments for all networks in a module	Save
RBK Remarks BacKup	backup copy of the remarks file	Save
FRC FoRCing	list of variables that can be forced, and their values	Save, if forced variables are designated
RTD Real Time Display	list of variables in the View Variables List	Edit view list
LST LiST	print file containing LDO, REM, and cross-reference, formatted for printing	Print
HEX HEXadecimal	hex representation of the LDO file	Hex-86 dump
BIN BINary	compiled (PiC900 language) LDO file - cannot be converted "back" into uncompiled state for animating, etc.	Download

Filename is name you assign		
SRV SeRVoSetup	setup data for all axes used in application	Save
PRO PROfile	ratio profile data for master/slave applications	Save
LIB LIBrary	user-defined functions/function blocks to be used in LDO	Make function
BAK BAcKup	backup copy of LIB file - the version created with the last save	Save
DPL DePendency List	list of all the files the currently loaded module depends upon	build dependency List
G&L Giddings & Lewis	compressed file from the Project Manager program	Update FMDSISK

Appendix B -PiCPro Function Errors and Error Codes

General Errors

For all functions: the output variables will have unpredictable values and the output at OK, DONE, or Q will not be energized when:

1. An output variable does not have enough bits to hold the result.
2. An output variable is an unsigned integer and the result is negative.
3. The operation attempts to divide a number by zero.
4. The input data is invalid.

I/O Function Block Error Codes

When the I/O functions execute, if an error occurs - the output at DONE is not energized, the output at FAIL is energized, and the output at ERR has one of the following values.

ERROR NUMBER : 01 ERROR NAME: OPCODE_ERROR

Invalid Device Open Mode specified for function (e.g.16#602).

ERROR NUMBER : 02 ERROR NAME: HANDLE_ERROR

This error is generated when the handle to a function is invalid. Some reasons for this might be - trying to use I/O functions with an unopened handle, trying to do a READ on a handle opened for WRITE ONLY, or trying to do a WRITE to a handle opened for READ ONLY.

ERROR NUMBER : 03 ERROR NAME: ALREADY_OPEN

Reserved for Giddings & Lewis. Internal GLOS Error.

ERROR NUMBER : 04 ERROR NAME: OPEN_ERROR

OPEN Mode (READ/WRITE/APPEND) specified is not valid.

ERROR NUMBER : 05 ERROR NAME: DEVICE_EMPTY

Reserved for Giddings & Lewis.

ERROR NUMBER : 06 ERROR NAME: READ_ERROR

This error can be one of the errors listed below depending on the device you are using.

1. DISK driver detects a checksum error on the DISK.
Media error on DISK reading disk FCB (File Control Block).
2. A parity, overrun, or framing error exists in the data in the input buffer of the COM port.

ERROR NUMBER : 07 ERROR NAME: WRITE_ERROR

Error writing data to the device.

ERROR NUMBER : 08 ERROR NAME: WRITE_PROTECT

Reserved for Giddings & Lewis.

ERROR NUMBER : 09 ERROR NAME: DEVICE_ERROR

The Seek origin is not A00, A01, or A02.

Improper device name - not PICPRO:, USER:, RAMDISK:.

Attempted to do a STATUS on a file.

Attempted to do a file operation on a non-DOS device.

Bad Parity in the configuration string or a hardware error in configuring the port.

Attempted to do a second or subsequent operation before the first one was completed.

ERROR NUMBER : 10 ERROR NAME: OUT_OF_HANDLES

Too many files open. A maximum of 10 handles can be used. Every READ-WRITE or APPEND operation uses 2 handles. A READ ONLY or a WRITE ONLY uses 1 handle.

ERROR NUMBER : 11 ERROR NAME: INVALID_DEVICE

Improper device name - not PICPRO:, USER:, RAMDISK:, or any name you have given a device at the NAMZ input of a function block.

ERROR NUMBER : 12 ERROR NAME: INVALID_FILE_NAME

Filename format is wrong (Bad filename).

Filename too large.

ERROR NUMBER : 13 ERROR NAME: TOO_MANY_DRIVERS

Reserved for Giddings & Lewis. Internal GLOS Error.

ERROR NUMBER : 14 ERROR NAME: TOO_MANY_CONNECTIONS

Reserved for Giddings & Lewis. Internal GLOS Error.

ERROR NUMBER : 15 ERROR NAME: DOS_READ_WRITE_ERROR

Error in Reading From or Writing to a DISK file.

ERROR NUMBER : 16 ERROR NAME: DOS_DEVICE_ERROR

Reserved for Giddings & Lewis.

ERROR NUMBER : 17 ERROR NAME: DOS_FILE_NOT_OPEN

READ, WRITE, or SEEK on a file that is not open.

ERROR NUMBER : 18 ERROR NAME: DOS_INVALID_ACCESS

Attempted to CLOSE a file that was not open.

SEEK has a problem seeking to a new location.

Attempted to write to a file opened for read only.

ERROR NUMBER : 19 ERROR NAME: DOS_FILE_NOT_IN_DIR

Filename not found in directory.

Volume name not found in directory.

ERROR NUMBER : 20 ERROR NAME: DOS_FILE_ALREADY_OPEN

Attempted to OPEN a file that is already open.

ERROR NUMBER : 21 ERROR NAME: DOS_WRITE_PROTECT

Error when trying to delete a READ ONLY file from DISK.

ERROR NUMBER : 22 ERROR NAME: DOS_MODE_ERROR

Invalid mode inputted to OPEN.

Invalid mode inputted to SEEK.

ERROR NUMBER : 23 ERROR NAME: DOS_OUT_OF_HANDLES

No more handles available - too many files open. A Maximum of 10 handles are available in the PiC900.

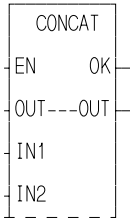
ERROR NUMBER : 24 ERROR NAME: DOS_END_OF_DIR

No more directory entries.

- ERROR NUMBER : 25 ERROR NAME: DOS_FUNCTION_LOCKED**
 Reserved for Giddings & Lewis. Internal GLOS error.
- ERROR NUMBER : 26 ERROR NAME: DOS_RENAME_NEW_EXISTS**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 27 ERROR NAME: DOS_RENAME_OLD_OPEN**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 28 ERROR NAME: DOS_RENAME_NO_OLD**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 29 ERROR NAME: DOS_ATTRIB_INVALID**
 Reserved for Giddings & Lewis. Internal GLOS Error.
- ERROR NUMBER : 30 ERROR NAME: DOS_HANDLE_TOO_LARGE**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 31 ERROR NAME: DOS_DISK_ERROR**
 General error on disk drive.
- ERROR NUMBER : 32 ERROR NAME: DEVICE_PARITY_ERROR**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 33 ERROR NAME: END_OF_DEVICES**
 Device driver for the device specified was not found.
- ERROR NUMBER : 34 ERROR NAME: IO_DEV_ABORTED**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 35 ERROR NAME: DUPLICATE_FILENAME**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 36 ERROR NAME: NO_MEMORY**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 37 ERROR NAME: DOS_NO_BUFFERS**
 No memory to get more file buffers.
- ERROR NUMBER : 38 ERROR NAME: DOS_DIR_NOT_EMPTY**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 39 ERROR NAME: DOS_DIR_NOT_FOUND**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 40 ERROR NAME: DOS_OUT_OF_FATS**
 No more entries available in the FAT (File Attribute Table). Reduce the number of files in this directory.
- ERROR NUMBER : 41 ERROR NAME: DOS_SDIR_EXISTS**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 42 ERROR NAME: FILESPEC_TOO_LONG**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 43 ERROR NAME: NO_DEFAULT_DEVICE**
 Reserved for Giddings & Lewis.
- ERROR NUMBER : 44 ERROR NAME: PARAMETER_ERROR**
 Occurs if the value entered at the CNT input of the READ function block is larger than the declared size of a string used as the BUFR input.

STRING Function Errors

When the STRING functions execute, if an error occurs - the output at OK is not energized and the STRING variable output will be null (have length zero).



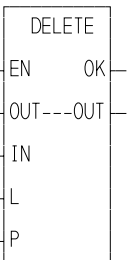
An error occurs:

If the length of IN1 > length of OUT

If the length of IN2 > length of OUT

If the length of IN1 + length of IN2 > length of OUT

If IN2, or IN3, ... or IN17 = OUT



An error occurs:

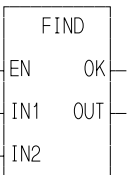
If P = 0

If P > 255

If P > length of IN

If L > 255

If length of IN - L > length of OUT

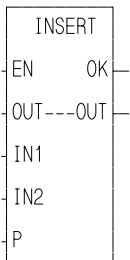


An error occurs:

If length of IN1 = 0

If length of IN2 = 0

If length of IN2 > length of IN1



An error occurs:

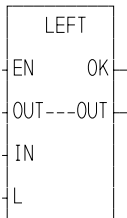
If P = 0

If P > 255

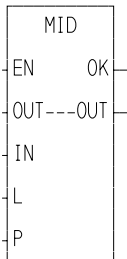
If P > length of IN1

If IN2 = OUT

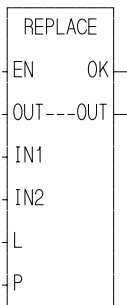
If length of IN1 + length of IN2 > length of OUT



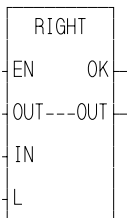
An error occurs:
 If $L > 255$
 If $L > \text{length of OUT}$



An error occurs:
 If $P > 0$
 If $P > 255$
 If $P > \text{length of IN}$
 If $L > 255$
 If $L > \text{length of OUT}$



An error occurs:
 If $P = 0$
 If $P > 255$
 If $P > \text{length of IN1}$
 If $L > 255$
 If $\text{IN1} = \text{OUT}$
 If $\text{IN2} = \text{OUT}$
 If $\text{length of IN1} - L + \text{length of IN2} > \text{length of OUT}$



An error occurs:
 If $L > \text{OUT}$
 If $L > 255$

NOTES

Appendix C - PiCServoPro Error Codes

E-stop, C-stop, programming, and timing errors

The servo codes covered in this appendix apply to the E-stop, C-stop, programming, and timing errors that can occur when using PiCServoPro. The first three apply to individual axes. The timing error is connected to the entire system.

1. C-stop (controlled-stop) errors

When a C-stop occurs, the following happens:

- The axis remains in servo lock and the axis is brought to a controlled stop at the rate specified by the controlled stop ramp in setup.
- The active and next queues are cleared.
- The FAST_QUE mode is canceled when the C-stop is reset.

2. E-stop (Emergency-stop) errors

When an E-stop occurs, the following happens:

- The system is out of servo lock.
- A zero voltage is sent to the analog outputs.
- The active and next queues are cleared.
- The FAST_QUE mode is canceled when the E-stop is reset.

If it is a loss of feedback E-stop error, then the machine reference must be redone.

In most respects, you are in a condition immediately following initialization with the exception of the que number. The que number does not start over but continues from where it left off when the E-stop occurred.

Remember that the que number is assigned by the software from 1 to 255. When 255 is reached, it rolls over to 1.

3. Programming errors

These errors occur during master/slave moves or a FAST_QUE call. They may prevent the move from being placed in the queue (or if the move is in the queue, abort the move) or they may prevent the OK on the function from being set.

4. Timing error

All the servo calculations for one interrupt must be completed in the time frame selected by you in setup (1, 2, 4, or 8 ms) before the next interrupt begins. If not, a timing error occurs.

The timing error is connected to the entire system. This error is monitored in the ladder program with the TME_ERR? function. If the boolean output at ERR is set, a timing error is occurring. Depending upon the system, this can affect performance.

IMPORTANT

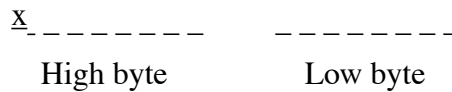
Always set an E-stop on all axes when a timing error occurs.

NOTE: The TME_ERR? function is an inquiry as to whether or not there has been a timing error. The C_STOP? and E_STOP? functions are inquiries as to whether a C-stop or E-stop has occurred.

These errors are accessible in one or two modes as described below.

1. The ERRS output on the E_ERRORS, C_ERRORS, P_ERRORS and the ERR output of TME_ERR? standard functions (Figure C-1)

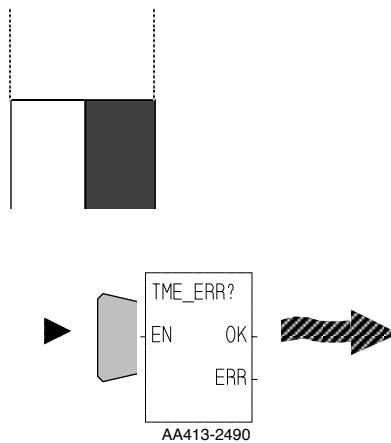
The ERRS output for E-stop, C-stop, and programming errors is a word, or two bytes, as shown below. The MSB bit (indicated by the “x”) in the high byte word indicates that there is an error in all cases. The low byte of the word is where the individual errors are located.



The ERR output for a timing error is a boolean.

The functions are used in your ladder program. By monitoring the ERRS output for each type of error, it is possible to tell what error is present by referring to the tables included here.

FIGURE C - 1. Standard Motion Functions with ERRS Word Output



2. The output side of the PiCTune screen in the Servo setup program (Figure C-2)

The E-stop, C-stop and programming errors are found at the bottom of the screen. An “E” appears in any of the eight bit locations if an error is present. The three tables that follow identify the errors.

NOTE

Pressing the <F9> Help key when in the PiCTune screen will bring up a description of all the E, C, and P errors. Use the <PgUp> and PgDn> keys to scroll through the categories.

The timing error is not available in this mode.

FIGURE C - 2. Errors on the PiCTune Screen

Axis	Fields	AXIS_1 Inputs	AXIS_1 Outputs
▶	Proportional Gain	100	Proportional Gain 101
▶	Integral Gain	0	Integral Gain 0
▶	Derivative Gain	0	Derivative Gain 0
▶	Analog Output Offset	0	Analog Output Offset 0
▶	Slow Speed Filter	0	Slow Speed Filter 0
▶	Feed Forward Percent	0	Feed Forward Percent 0
			Following Error 0
			Commanded Velocity 0
			Feedback Position 0
			Commanded Position 0
			Analog output in mv 0
			Active Move None [000]
			Next Move None [000]
			EStop Error
			CStop Error
			Prog Error Low Byte
			Prog Error High Byte

The errors connected with each bit location are covered in the tables that follow.

In these tables, the **Bit Location** column indicates which bit is set in the low or high byte of the word connected to each error.

The **Hex Value** column represents the form the error is returned in while monitoring the ERR output in the ladder program.

Emergency Stop Errors

Error	Description	Bit Location (low byte)								Hex Value (in LDO)
		8	7	6	5	4	3	2	1	
(not used)										
(not used)										
(not used)										
(not used)										
User-set	An E-stop on a servo axis has occurred which was called in the ladder using the ESTOP function.					E				8008
Overflow error	<p>A slave delta overflow during runtime has occurred. This problem is most likely to occur if you are moving at a high rate of speed and/or the slave distance is very large compared to the master distance.</p> <p>There are two conditions that can set this bit.</p> <ol style="list-style-type: none"> 1. In FU, if the master moved position times the slave distance entered is greater than 31 bits. 2. In FU, if the $\frac{\text{master moved} \times \text{SDIS}}{\text{MDIS}}$ is greater than 16 bits. 						E		8004	
Excess error	When an excess following error has occurred, the axis has exceeded the limit entered in the Servo setup program as the following error limit. This represents the maximum distance the commanded axis position can be from the actual axis position.							E		8002
Loss of feedback	A loss of feedback from the feedback device has occurred. Available for servo and digitizing axes.								E	8001

Controlled Stop Errors

Error	Description	Bit Location (low byte)								Hex Value (in LDO)
		8	7	6	5	4	3	2	1	
Part reference error	Move was in progress when a part reference or a part clear function was called.	E								8080
Part reference dimension error	When the dimension for the part reference was converted to feedback units, it was too big to fit into 29 bits.		E							8040
Distance or position move dimension error	When the dimension for the move was converted to feedback units, it was too big to fit into 31 bits.			E						8020
Feedrate error*	When the feedrate for the move was converted to feedback units per servo up-grade, it was too big to fit into 32 bits or it exceeds the velocity limit entered in setup.				E					8010
Machine reference dimension error	When the dimension for the machine reference was converted to feedback units, it was too big to fit into 29 bits.					E				8008
User-defined C-stop	When this bit is set, a user-defined C-stop has occurred.						E			8004
Negative software limit exceeded	The command position exceeded the user-defined negative software end limit.							E		8002
Positive software limit exceeded	The command position exceeded the user defined positive software end limit.								E	8001

*This error can occur with feedrate override, new feedrate, position, distance, velocity, or machine reference moves.

The programming errors listed in the next two tables can be divided into two categories--those connected to the FAST_QUE function and those connected to the master/slave moves.

The first error listed (bit location 8 of low byte) is connected to the FAST_QUE function. The remaining errors are connected to the master/slave moves.

Programming Errors (Low Byte)

Error	Description	Bit Location (low byte)								Hex Value (in LDO)	
		8	7	6	5	4	3	2	1		
The FAST axis in the FAST_QUE function moved too far in wrong direction	The axis traveled more than 65,535 FU in the opposite direction of the value entered in DIST of the FAST_QUE function.	E									8080
Profile number not found	Data for a profile move is not valid.		E								8040
Master axis not available	This error can occur when using the FAST_QUE function or the functions for master/slave moves (RATIO_GR, RATIOSYN, or RATIOPRO). The conditions that can set this bit: 1. Master axis or fast axis not initialized 2. Interrupt rates different for axes 3. Axis at slave input is the same as axis at master input in master/slave moves			E							8020
(not used)											
(not used)											
(not used)											
(not used)											
Master start position for lock on	When the dimension for the lock position was converted to feedback units, it was too big to fit into 32 bits.									E	8001

Programming Errors (High Byte)

Error	Description	Bit Location (high byte)								Hex Value (in LDO)
		8	7	6	5	4	3	2	1	
	This bit is set whenever any of the remaining 15 bits is set.	X								8000
(not used)										
(not used)										
(not used)										
Master axis beyond start point	The master axis is beyond its starting point for a ratio synchronization (RATIOSYN) move.					E				8800
Slave axis beyond start point	The slave axis is beyond its starting point for a ratio synchronization (RATIOSYN) move.						E			8400
Master distance not valid	When the master distance is converted to feedback units, it is greater than 16 bits.							E		8200
Slave distance not valid	When the slave distance is converted to feedback units, it is greater than 16 bits.								E	8100

 STARTING
PiCPro/PiCServoPro

NOTES

Appendix D - Function Input/Output Data Type Reference

This is a quick reference giving the data types of all the inputs and outputs of the functions and function blocks used in PiCPro and PiCServoPro. The input or output is on the left and the data type is on the right.

If an input/output is listed more than once, it has more than one data type depending on the function it comes from. In that case, the functions are listed below the input/output and data type. All the various IN inputs and OUT outputs are covered in Tables D-1 and D-2.

4mA0.....	BOOL	D	PID
10ms.....	BOOL	DABL.....	ERR
100ms	BOOL	DATA	A_INCHIT, A_INCHRD,
A		DAY.....	A_INMDIT, ARTCHIT,
ACCL.....	UDINT	DBUF.....	ARTDCHRD, ARTDMDIT,
ACT	DINT	DCNT	ATMPCHIT, ATMPCHRD,
PID		DECL	ATMPMDIT, STRTSERV,
ACT	INT	DEN.....	CAPTINIT, SERVOCLK
NETRCV, NETSND, READ, WRITE		DEST	ERRS.....
ACTV.....	WORD	DID	ESTP
ANGL	REAL/LREAL	DIFF	ET
AXIS	USINT	DIM.....	F
B		DIR.....	FAHR.....
BEG.....	BOOL	DIR.....	FAIL.....
BIPO.....	BOOL	DIST	FAST.....
BKPR	BOOL	DIST	G
BTVL	DINT	REGIST	G
BUFR.....	ARRAY, STRUCT, STRING	DVND.....	H
C		DONE.....	HILT
CAM	ARRAY OF STRUCTURES	DROP.....	HNDL.....
CD	BOOL	DTST.....	I
CFGZ	STRING	DVSR.....	IGNR.....
CHAN	USINT	as DVND or DINT if DVND = TIME	INPS.....
CLRC	UINT	E	INs.....
CLSD.....	BOOL	ELEM	IST
CMD	UINT	EN	K
CNT	INT	ERR.....	K.....
COMN	STRUCT	TME_ERR?, ANLG_OUT	L
CONF	STRUCT	ERR.....	L
COS.....	REAL/LREAL	ASSIGN, CLOSE, CONFIG,	LD
CSTOP.....	BOOL	NETOPEN, NETRCV, NETSND,	LEN.....
CNTL.....	UINT	OPEN, READ, SEEK, STATUS,	LGTH.....
CU	BOOL	WRITE, DELFIL, FRESpace,	LOG
CV	INT	RENAME, COORD2RL,	LOLT.....
		TUNEWRIT	LN
		ERR.....	REAL/LREAL

INPUT/OUTPUT
DATA TYPES

M	PLUS..... BOOL	SDST..... DINT
MAN..... BOOL	PNT..... USINT	SEG1..... STRUCT
MAST..... USINT	PNUM..... USINT	SET..... BOOL
MAX..... same as MIN	POS..... DINT	SID..... USINT
MCND..... NUMERIC or TIME	PROD..... same as MCDN	SIN..... REAL/LREAL
MDST..... DINT	PT..... TIME	SIZE..... DINT
μ sec..... UINT	PV..... INT	SIZE..... USINT
MIN..... any	Q	CAPTINIT
MODE..... INT	Q..... BOOL	SLOT..... USINT
MOVE..... STRUCT	QAVL..... BOOL	SLPE..... ARRAY OF STRUCTURES
MPLR..... same NUMERIC as MCND or DINT if MCND = TIME	QTY..... DINT	SPT..... DINT
MSTR..... DINT	QUE..... USINT	SQR..... UDINT, UINT, USINT, or constant
N	QUOT..... same as DVND	SRCE..... ARRAY OF STRUCT
N..... USINT	R	SSTR..... DINT
NAME..... STRING	R..... BOOL	STAT..... INT
NAMZ..... STRING	RACK..... USINT	NETMON, STATUS
NUM..... INT	RATE..... UDINT	STAT..... WORD
RATIOSCL	RATE..... TIME	STATUSSV
NUM..... NUMERIC	SERVOCLK	STR..... STRING
NUM2STR, STR2NUM	RDNE..... BOOL	STRC..... STRUCT
NUM..... REAL/LREAL	REAL..... ARRAY OF STRUCT	SUM..... same as IN1
EXP, LOG, LN	REFD..... DINT	T
NUM..... USINT	REM..... same as DVND	TAN..... REAL/LREAL
USIN2STR, STR2USI	REQ..... BOOL	TBUF..... ARRAY, STRUCT, STRING
O	REV..... BOOL	TCNT..... INT
OFF..... DINT	ROOT..... same as SQR	TIME..... UINT
OFST..... UINT	RNGE..... USINT	TOLR..... UDINT
OK..... BOOL	RPER..... USINT	TYPE..... USINT
ON..... DINT	RPTP..... BOOL	V
OPTN..... WORD	RSCD..... STRUCT	VALU..... INT
ORG..... INT	RSLT..... DINT	VAR..... SINT
OUTs..... (See Table D-2.)	RVAL..... BOOL	VARS..... STRUCT
P	S	W
P..... NUMERIC	SDIR..... BOOL	WEEK..... BOOL
DELETE, INSERT, MID, REPLACE		

TABLE D-1. IN inputs

[Inputs for extensible functions are followed with (ext).

Input	Data type	Functions
IN	BITWISE	NOT, ROL, ROR, SHL, SHR
	BOOL	TOF, TON, TP
	DATE	DATE2STR
	DATE_AND_TIME	CLOCK, DT2DATE, DT2STR, DT_2_TOD
	NUMERIC	ABS, NEG
	STRING	DELETE, LEFT, MID, RIGHT
	TIME	TIME2STR
	TIME_OF_DAY	TOD2STR
	same as MIN	LIMIT
IN0	NUMERIC or TIME	SUB
	any except STRUCT	SEL
IN0 (ext)	any except STRUCT	MUX
IN1	any except BOOL or STRUCT	NE
	DATE	D_TOD2DT, S_D_D
	DATE_AND_TIME	A_DT_T, S_DT_DT, S_DT_T
	STRING	FIND, INSERT, REPLACE
	TIME_OF_DAY	A_TOD_T, S_TOD_T, S_TOD_TO
	same as IN0	SEL, SUB
IN1 (ext)	any	MOVE
	any except BOOL or STRUCT	EQ, GE, GT, LE, LT, MAX, MIN
	BITWISE	AND, OR, XOR
	NUMERIC or TIME	ADD
	STRING	CONCAT
	same as IN0	MUX
IN2	DATE	S_D_D
	DATE_AND_TIME	S_DT_DT
	TIME	A_DT_T, A_TOD_T, S_DT_T, S_TOD_T
	TIME_OF_DAY	D_TOD2DT, S_TOD_TO
	same as IN1	NE
	STRING	FIND, INSERT, REPLACE
IN2 (ext)	same as IN1	ADD, AND, EQ, GE, GT, LE, MAX, MIN, OR, XOR
	STRING	CONCAT, REPLACE

INPUT/OUTPUT
DATA TYPES

TABLE D-2. OUT outputs

Output	Data type	Functions
OUT	BOOL	CAM_OUT, EQ, GE, GT, LE, LT, NE
	DATE	DT2DATE
	DATE_AND_TIME	A_DT_T, CLOCK, D_TOD2DT, S_DT_T
	same as IN	ABS, NEG, NOT, ROL, ROR, SHL, SHR
	same as IN0	MUX, SEL
	same as IN1	AND, OR, XOR
	same as MIN	LIMIT
	NUMERIC	FIND
	TIME	S_D_D, S_DT_DT, S_TOD_TO
	TIME_OF_DAY	A_TOD_T, DT2TOD, S_TOD_T
OUT1	same as IN1	MAX, MIN, MOVE
OUT---OUT	STRING	CONCAT, DATE2STR, DELETE, DT2STR, INSERT, LEFT, MID, REPLACE, RIGHT, TIME2STR, TOD2STR,

NOTE

There are also INs/OUTs on all the data type conversion functions. In those functions, the IN is always the data type you are converting from and the OUT is always the data type you are converting to.

Appendix E - PiCServoPro Reference Card - Errors/Variables

#	STRTSERV func errs
0	No error
1	Bad user function data
2	Not enough low memory
3	Feedback module(s) not found
4	Analog module(s) not found

Word output from STATUSV function		
Characteristic	Binary value	Hex
Move started	00000000 0000000(1)	0001
Fast input occurred	00000000 000000(1)0	0002
Fast input on	00000000 00000(1)00	0004
Good mark detected	00000000 0000(1)000	0008
Bad mark detected	00000000 000(1)0000	0010
DIST + TOLR exceeded	00000000 00(1)00000	0020
Fast input rising	00000000 0(1)000000	0040

E-stop Error	Bit Location (low byte)								Hex Value (in LDO)
	8	7	6	5	4	3	2	1	
User-set					E				8008
Overflow error						E			8004
Excess error							E		8002
Loss of feedback								E	8001

C-stop Error	Bit Location (low byte)								Hex Value (in LDO)
	8	7	6	5	4	3	2	1	
Part reference error	E								8080
Part reference dimension error		E							8040
Distance or position move dimension error			E						8020
Feedrate error				E					8010
Machine reference dimension error					E				8008
User-defined C-stop						E			8004
Negative software limit exceeded							E		8002
Positive software limit exceeded								E	8001

Programming Error	Bit Location (high byte)								Hex Value (in LDO)
	8	7	6	5	4	3	2	1	
Set whenever a P error occurs.	X								8000
Master axis beyond start point					E				8800
Slave axis beyond start point						E			8400
Master distance not valid							E		8200
Slave distance not valid								E	8100

Programming Error	Bit Location (low byte)								Hex Value (in LDO)
	8	7	6	5	4	3	2	1	
The FAST axis in the FAST_QUE function moved too far in wrong direction	E								8080
Profile number not found		E							8040
Master axis not available			E						8020
Master start position for lock on								E	8001

REFERENCE CARD

Variables used with the READ_SV (**Read** column) and WRITE_SV (**Write** column) functions. These variables are used with servo (**S**), time (**T**), and/or digitizing (**D**) axes.

V#	Variable Description	Read	Write
1	Actual position	S, T, D	T
2	Move type 11 pos 18 ratiopro 12 dist 20 ratiosyn/gr 14 vel st 22 ratiocam 16 lad ref or 23 ratioslp fast ref 24 ratio real	S	
3	Command position	S, D	
4	Position error	S	
5	Slow velocity filter error	S	
6	Command velocity	S, T	T
7	Position change	S, D	
8	Feedback last	S, D	
9	Fast input position	S, D	
10	Reg/ref position change	S, D	
11	Consecutive bad marks	S, D	S, D
12	Rollover on position	S, T, D	S, T, D
13	Slave offset incremental	S	S
14	Master offset incremental	S	S
15	Slave offset absolute	S	S
16	Master offset absolute	S	S
17	Slave offset filter		S
18	Master offset filter		S
19	Fast input direction		S, D
20	Fast input distance	S, D	
21	Reversal not allowed	S	S
22	Fast input position (software)	S, D	
23	Position (software) of axis 1 with fast input on axis 2	S, D	S, D
24	Registration switch	S, D	S, D

V#	Variable Description	Read	Write
25	Fast queuing	S	S
26	Synchronized slave start	S, D, T	S, D, T
27	Backlash compensation	S	S
28	TTL feedback	S, D	S, D
29	Reference switch position	S	
30	Filter time constant	S	S
31	Filter error limit	S	S
32	Velocity compensation flag	S	S
33	Filter lag	S	
34	Position change over several interrupts	S, D	S, D
35	Part reference offset	S, D	
36	Software upper limit	S	S
37	Software lower limit	S	S
38	Commanded position (before slow velocity filter)	S, D	
39	Following error limit	S	S
40	In-position band	S	S
41	Current segment number	S	
42	Slave distance into segment	S	
43	Master distance into segment	S	
44	Set user iteration command	S	S
45	User iteration command	S	S
46	Set user PID command	S	S
47	User PID command	S	S
48	Disable servo software	S	S
49	(Reserved)		
50	Override endlimit check	S	S
51	Write SERCOS position	S	S

Data Capture

Axis variables that can be captured on a servo interrupt basis with the CAPTINIT function.

VAR	Description	Type	VAR	Description	Type
1	Actual Position	DINT	7	Position change	INT
2	Fast Input	BYTE	8	Feedback position	DINT
3	Commanded position	DINT	9	Prefilter commanded	DINT
4	Position error	DINT	10	Prefilter command change	INT
5	Slow velocity filter error	INT	11	Remaining master offset	DINT
6	Command change	INT	12	Remaining slave offset	DINT

Error numbers and descriptions for the CAPTINIT function ERR output are listed below.

0	No error
1	The CAPTSTAT function has not stopped capturing data from a previous data capture initialization.
2	An axis number in the structure is invalid.
3	The limit of eight variables in the array of structures has been exceeded.
4	Parameter number in the structure is out of range.
5	The CAPINIT function was called before the STRTSERV function was called.

NOTES

Appendix F - Stepper Reference Card

#	Profile Commands	Range
1	Distance move	±2,147,352,575 steps
2	Position move	±2,147,352,575 steps
3	Velocity move	±1,000,000 steps/sec
4	Set maximum velocity	1-1,000,000 steps/sec
5	Set acc/dec rate	1-16,777,215 steps/sec/sec
6	Set reference	±2,147,352,575 steps
7	Pause	N/A

#	Control Words
1	Enable profile
2	Pause profile
3	Continue profile
4	E-stop
5	C-stop
6	Step/direction mode (default)
7	CW/CCW mode

Word output from STEPSTAT function			
Characteristic	Binary value	Decimal	Hex
Profile enabled	00000000 0000000x(1)	1	0001
Profile paused	00000000 000000x(1)0	2	0002
At velocity	00000000 00000x(1)00	4	0004
Que empty	00000000 0000x(1)000	8	0008
Que full	00000000 000x(1)0000	16	0010
Control word not processed	00000000 00x(1)00000	32	0020

Errors displayed in the .ERROR member of stepper structure	
Error	#
No error	0
Invalid rack number or remote rack not available	1
Invalid slot number	2
Module not found at rack and slot location or not enough channels on module	3
Invalid command number	4
Invalid data for the command	5
Invalid control number	6
Stepper function called before STEPINIT function called	7

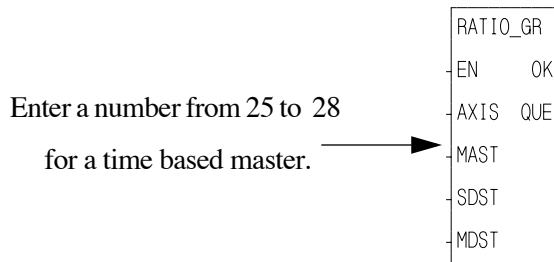
STEPPER
REFERENCE CARD

NOTES

Appendix G - Time Axis

The time axis feature allows a servo axis to be slaved to time instead of a physical master position transducer. All the master/slave functions can be used with a time axis.

There are four axis numbers reserved for a time based master; 25, 26, 27 and 28. This is the number used to identify the master time axis on the input to a master/slave function as shown below.



The S_CURVE function or the command velocity variable 6 can be used to move a time axis. The time axis can be manipulated with variables 1, 12, and 26 as described below. Use the WRITE_SV and READ_SV functions to work with these variables.

To Reference: Actual Position (Variable 1)

The actual position variable allows you to read the position of the time axis or change the current position by writing a value with the WRITE_SV function. The variable is in ladder units.

Range = +2147483647 to -2147836648 ladder units

To Control Velocity: S_CURVE Function or Command Velocity (Variable 6)

You can use either the S_CURVE function or the command velocity variable 6 to move a time axis.

S_CURVE Function

When using the S_CURVE function with a time axis, you can use the distance, position, or velocity moves to "move" the axis. The S_CURVE function must be called first when using these moves. See the S_CURVE description in the Function/Function Block Reference Guide.

Command Velocity Variable 6

If you are not using the S_CURVE function, the command velocity variable can be used to define how fast the time axis will travel. It is programmed in ladder units per second. When the WRITE_SV function is called with variable 6, the time axis will step to the programmed velocity.

Each ladder unit of the time axis travel represents the portion of time programmed by this variable. For example, if a value of 1000 is programmed as the number of ladder units per second for the velocity, then the time axis would move one ladder unit in 1 msec. If the master distance (MDST) was set at 1000 and the slave distance (SDST) was set at 2000 in the `RATIO_GR` function above, it would take the slave axis 1 sec to move 2000 units.

By entering a zero, the time axis is stopped. In effect, you have "stopped time." This provides the ability to synchronize multiple slave axes. You call all the moves you want to synchronize and then write a non zero value to variable 6. All the axes will begin motion at the same time.

NOTE: In order for all slave axes to start at the same time, the master start position of any master/slave move with a MSTR input would have to have the same value (or zero) at its MSTR input. If the option to ignore master start is selected, the slave axes will start when the master axis begins to move.

An alternative method for synchronizing slave starts is to use variable 26 as described below.

Range = $\pm 2,000,000$ ladder units/sec

To Define Rollover: Rollover on Position (Variable 12)

The rollover on position variable allows you to select where the time will reset to zero. The variable is entered in ladder units.

NOTE: Without rollover on position when 2,147,483,647 is reached, the next number will be -2,147,483,648. The count continues to zero and back up to 2,147,483,647, etc.

Range = 1 to 536,870,912 ladder units (Entering a zero turns rollover on position off.)

To Synchronize Slave Axes: Synchronized Slave Start (Variable 26)

The synchronized slave start variable allows you to tell the time axis which of its slave axes must be queued up before any of them begin their move. Each slave axis you want to synchronize is identified by setting a bit in a DINT using the lower 16 bits where the LDB = axis 1 and the MSB = axis 16. When the last "set" axis has been queued, all the slave axes will begin their move on the next interrupt.

The `WRITE_SV` function with variable 26 must be called before the move. It can be called again when you want to identify a different set of synchronized slave axes. Change the bits only after the slave axes identified in the first `WRITE_SV` function have started to move.

Writing a zero to variable 26 clears all identified axes. The `READ_SV` function can be used to read the number of slave axes being synchronized.

Appendix H - Command Line Switches

The following list describes MS-DOS command line switches that are available for use with PiCPro in certain situations. Type in the switch on the command line after typing *picpro* or *picservo*. If you are using more than one switch on the command line, each switch must be separated by a space.

Command	Description
/e	Extended/Expanded Memory Inhibits PiCPro from using extended/expanded memory. Use this switch if PiCPro is having problems with the EMM (Extended Memory Manager) currently installed in your workstation.
/f	Soft Bit Memory Enables soft bit memory in all Turbo processors that do not have soft bit memory as standard. NOTE: Soft bit memory is standard in the PiC91X processors.
/u	Update UDFBs This switch <i>must</i> be used so that <i>all</i> UDFBs will be updated the first time you run PiCPro software after loading the latest version of PiCPro.

The MS-DOS user interrupt number 65h is used as the default interrupt by the Giddings & Lewis Communications driver to communicate to the PiC control. If this user interrupt is used by some other program in your PC, you can change to another user interrupt number by using the */i* switch. Other user interrupt numbers available to enter at <num> include 60, 61, 62, 63, 64, or 66.

Command	Description
/i<num>	Interrupt Number Allows you to change the default interrupt number (65) used for communications.

NOTES

Appendix I - Diagnostic LED Error Codes

While the PiC900 is running, the DIAG LED on the CPU module will flash a three digit code signal if there is an error. For example, if there is a long pause-flash-pause-flash-flash-pause-flash-flash-flash-long pause, the code is 123. The errors are described below

Code	Error	Description
122	No math coprocessor	Attempted to perform floating point operation with no math coprocessor installed on the CPU.
123	Scan too long	A ladder scan loss has occurred because the CPU takes more than 200 ms to scan the application program. Whenever the scan light is out, the discrete outputs go to the OFF state and the analog outputs are zeroed.
124	Excessive overhead	The system overhead update time is excessive.
125	Insufficient memory	There is insufficient memory on the CPU to run the current program.
126	No hardware bit memory	There is no bit memory installed on the CPU and the program requires it.
127	No software bit memory	There is no bit memory capability via software and the program requires it.
222	Driver error	No driver support on the CPU for the I/O module. Update your system EPROMs.
22_	Master rack error	The I/O modules in the master rack do not match what was declared in the hardware master declaration table. The number of flashes in the third digit () identifies the slot number that is in error.
231*	No daughter board	There is no communications daughter board installed on the CPU when attempting to do expansion I/O communications.
232*	Communications error	A failure has occurred in remote I/O communications.
233*	Number of racks error	The number of expansion racks in the system does not match the number of expansion racks declared in the expansion hardware declaration table.
3__*	Expansion rack error	The I/O modules in the expansion rack(s) or the block I/O modules do not match what was declared in the expansion hardware declaration table. For rack expansion: The number of flashes in the second digit indicates the remote rack (1 through 8). The number of flashes in the third digit indicates the slot number. For block I/O modules: The number of flashes in the second and third digits indicates the block I/O module (01 through 77). The second digit will flash a 1 - 7, 10 for 0.. The third digit will flash a 1 - 9, 10 for 0. For example, if the second digit flashes 3 times and the third digit flashes 10 times, the module is 30 .

LED ERROR CODES

* Errors connected with I/O expansion. Refer to the I/O Driver Module write-up in the Hardware Manual for more information.

NOTES

Appendix J - IBM ASCII Chart

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
00	0x00	NUL	32	0x20	SPC	64	0x40	@	96	0x60	`
01	0x01	☺	33	0x21	!	65	0x41	A	97	0x61	a
02	0x02	☹	34	0x22	"	66	0x42	B	98	0x62	b
03	0x03	©	35	0x23	#	67	0x43	C	99	0x63	c
04	0x04	®	36	0x24	\$	68	0x44	D	100	0x64	d
05	0x05	β	37	0x25	%	69	0x45	E	101	0x65	e
06	0x06	™	38	0x26	&	70	0x46	F	102	0x66	f
07	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
08	0x08	BS	40	0x28	(72	0x48	H	104	0x68	h
09	0x09	HT	41	0x29)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	♪	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	j	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	~	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	∨	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	↑↓	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	!!	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	ƒ	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	§	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	-	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	◊	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	≠	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	∅	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	Æ	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	¨	59	0x3B	;	91	0x5B	[123	0x7B	{
28	0x1C	¿	60	0x3C	<	92	0x5C	\	124	0x7C	:
29	0x1D	↔	61	0x3D	=	93	0x5D]	125	0x7D	}
30	0x1E	s	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	t	63	0x3F	?	95	0x5F	_	127	0x7F	>

ASCII CHART

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	0x80	Ç	160	0xA0	à	192	0xC0	ı	224	0xE0	‡
129	0x81	ü	161	0xA1	í	193	0xC1	ı	225	0xE1	·
130	0x82	é	162	0xA2	ó	194	0xC2	¬	226	0xE2	,
131	0x83	â	163	0xA3	ú	195	0xC3	√	227	0xE3	„
132	0x84	ä	164	0xA4	ñ	196	0xC4	f	228	0xE4	‰
133	0x85	à	165	0xA5	Ñ	197	0xC5	≈	229	0xE5	Â
134	0x86	å	166	0xA6	ª	198	0xC6	Δ	230	0xE6	Ê
135	0x87	ç	167	0xA7	º	199	0xC7	«	231	0xE7	Á
136	0x88	ê	168	0xA8	ı	200	0xC8	»	232	0xE8	Ë
137	0x89	ë	169	0xA9	©	201	0xC9	...	233	0xE9	Ë
138	0x8A	è	170	0xAA	™	202	0xCA		234	0xEA	Ω
139	0x8B	ï	171	0xAB	´	203	0xCB	À	235	0xEB	Î
140	0x8C	î	172	0xAC	¨	204	0xCC	Ã	236	0xEC	∞
141	0x8D	ì	173	0xAD	ı	205	0xCD	Õ	237	0xED	Δ
142	0x8E	Ä	174	0xAE	«	206	0xCE	Œ	238	0xEE	Œ
143	0x8F	Å	175	0xAF	»	207	0xCF	œ	239	0xEF	«
144	0x90	É	176	0xB0	∞	208	0xD0	–	240	0xF0	‡
145	0x91	Æ	177	0xB1	±	209	0xD1	—	241	0xF1	±
146	0x92	Æ	178	0xB2	≤	210	0xD2	“	242	0xF2	≥
147	0x93	ô	179	0xB3	≥	211	0xD3	”	243	0xF3	≤
148	0x94	ö	180	0xB4	¥	212	0xD4	‘	244	0xF4	Û
149	0x95	ó	181	0xB5	μ	213	0xD5	’	245	0xF5	ı
150	0x96	û	182	0xB6	∂	214	0xD6	÷	246	0xF6	∏
151	0x97	ù	183	0xB7	Σ	215	0xD7	◊	247	0xF7	≈
152	0x98	ÿ	184	0xB8	∏	216	0xD8	ÿ	248	0xF8	º
153	0x99	Ö	185	0xB9	π	217	0xD9	ÿ	249	0xF9	•
154	0x9A	Ü	186	0xBA	‡	218	0xDA	/	250	0xFA	·
155	0x9B	ç	187	0xBB	ª	219	0xDB	α	251	0xFB	√
156	0x9C	£	188	0xBC	º	220	0xDC	<	252	0xFC	,
157	0x9D	¥	189	0xBD	Ω	221	0xDD	>	253	0xFD	”
158	0x9E	û	190	0xBE	æ	222	0xDE	fi	254	0xFE	¸
159	0x9F	ü	191	0xBF	ø	223	0xDF	fl	255	0xFF	

Application Note

Application Note 1

Reading and Writing STRINGS from a Structure

The following applies only to Reading and Writing STRINGS that are members of a Structure *using just the structure name* as the **Memory Area (BUFR)** input to the READ and WRITE function blocks in PiCPro.

Any variable of type 'STRING' has an actual size that is 2 more than its 'Declared' length. These two bytes are in the beginning of the STRING and are normally hidden from the user. The first byte contains the maximum or 'Declared' length of the STRING, and the second byte contains the actual length of the STRING.

This means that the structure size will be increased by two bytes for every STRING element in the structure.

Examples:

```
1. MY_STRUC    STRUCT
    .ONE       DINT
    .NAME      STRING[10]
    .TWO       DINT
                END_STRUCT
```

Example 1 has a size of 20 bytes (4 bytes for .ONE, 12 bytes for the .NAME element and 4 bytes for .TWO).

```
2. MY_STR2     STRUCT
    .NAMES     STRING[10](0..3)
                END_STRUCT
```

Example 2 has a size of 48 bytes (12 bytes for each STRING of 10, and it is an array of 4 STRINGS).

So if you Write a structure that contains a 'STRING' type member, **it is important** to write out the complete structure (i.e. if you write out MY_STR2 make sure you write out 48 bytes and not 40 bytes).

Similarly if you Read into a structure that contains a STRING type member, **it is important** to read in the complete structure (i.e. if you read in MY_STR2 make sure you read in 48 bytes and not 40 bytes).

This does not apply if you Read from/Write to a STRING type variable itself. In the example above, it would not apply if you read from/write to MY_STRUC.NAME or MY_STR2.NAMES(3). In this case, PiCPro automatically updates the two hidden length bytes as the STRING is read in.

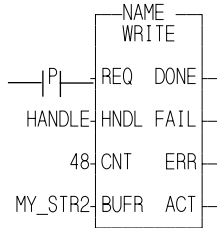
Some additional examples:

1. Consider a ladder with a structure declared as :

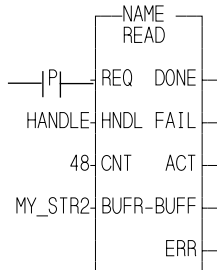
```

MY_STR2  STRUCT
.NAMES   STRING[10](0..3)
          END_STRUCT
  
```

To write out all four STRINGS in the structure completely, you would use,



To read all four STRINGS into the structure completely, you would use,



Note that this also means that the device you are reading the STRINGS from in this manner **MUST** store them in the following format:

1 BYTE 'Declared' Length in PiCPro (*In this example 10*)

1 BYTE Actual Length of the STRING (actual number of characters in this STRING)

(The actual length must be less than or equal to the declared length)

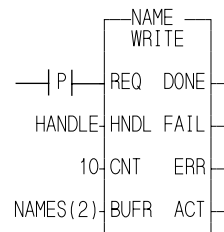
BYTES of the STRING itself. (*In this example 10 Bytes. Even if the actual number of characters in the STRING is less than the Declared Length, **the STRING must be padded up to the declared length.** You can use any character you choose to pad the STRING.*)

If you are reading from/writing to the STRING itself, **you do not** need to account for the two hidden bytes as in the examples below. PiCPro will automatically update these bytes when it knows the input to BUFR is a 'STRING'.

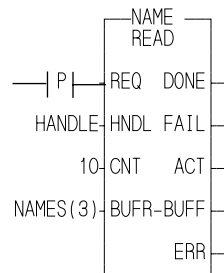
2. Consider a STRING declared as :

NAMES STRING[10](0..3)

To write out just one STRING completely, you would use,



To read in the NAME written above, you would use,



NOTES

Index

A

- acceleration ramp 6-17
- aliases 4-14
- amplifier telegram (AT) 10-3
- analog input 6-7
 - feedback 6-7
 - filter times 6-7
 - ranges 6-7
- analog output
 - offset 6-19
 - polarity 6-19
- animate 2-7
- Animate, Monitor menu
 - real time display 4-5
- Application Note N-3
- application notes
 - SERCOS 10-17
- application program
 - Download 4-2
 - print ladder 4-13
 - test on Download 4-4
- Arithmetic functions 3-4
- arrays 3-16
 - declare 3-34
 - define 3-34
 - elements 3-34
 - initialize 3-34
 - of structs w/arrays 3-43
 - of structures 3-41
 - size 3-34
- ASCII 3-13
 - chart J-1
- AT
 - structure 10-4
- attributes
 - UDFB 3-10, 8-9
 - variables 3-27
- Autoexec.bat 1-8
- axis profile setup *see PiC Profile*
- axis units, define 6-1, 6-15
- axis (in setup)
 - label 6-11
 - number 6-11

B

- Backup

- application program 4-2
 - Processor menu, define 4-26
- baud rate 1-15
- Bin-86 dump 4-18
- binary
 - file 4-18
 - numbers 3-32
- Binary functions 3-4
- bit 3-11
- Bitwise
 - constants 3-11
 - data types 3-11
 - initial value 3-32
- blown fuse 3-31
- BOOL boolean 3-11
- boolean
 - coils 3-3
 - contacts 3-3
 - control relays 3-3
 - elements 3-3
 - I/O Pt. 3-31
 - logic 3-3
 - structure member 3-37
- BYTE 3-11

C

- carriage return 3-13
- CD 3-7
- cells 2-1
- channel 3-30
- character codes 3-13
- clear
 - column 3-19
 - row 3-19
- close module 2-17
- Closed loop *see position loop*
 - define 5-1
- codes
 - diagnostic I-1
 - floating point 4-7
- coils
 - add to network 2-15
 - boolean 3-3, 3-11
 - define 2-3
 - find 3-50
 - latch 3-3
 - reset 3-3

- set 3-3
- unlatch 3-3
- color display 1-14
- column
 - clear 3-19
 - delete 3-19
 - insert 3-19
- commanded voltage output scaling 6-19
- comments 2-3, 3-48
- Config.sys 1-9, 7-10
- connect to node, Processor menu 4-28
- connect, input/output 3-7
- constants 3-10, 3-11
- contacts
 - add to network 2-15
 - define 2-3
 - transitional 3-3
- control relays
 - boolean 3-11
 - define 3-3
- controlled stop ramp 6-17
- copy
 - axis setup data 6-20
 - network 3-48
- Counter function blocks 3-4
- counts/motor revolution, output scaling 6-16
- C-stop
 - define C-1
 - errors C-5
 - bit locations C-5
 - hex value C-5
- CU 3-7
- Cut and paste within network 2-16
- cyclic data
 - SERCOS 10-9
- C_ERRORS function C-2
- C_STOP? function C-2

D

- Data In 3-10
- Data inVerted 3-10
- Data Out 3-10
- data types
 - Bitwise 3-11
 - DATE 3-14
 - DATE_AND_TIME 3-14
 - define 3-10
 - input to function 3-6
 - inputs/outputs D-1
 - Numeric 3-12

- String 3-13
- Time duration 3-14
- Time of day 3-14
- Type field 3-29
- Datatype functions 3-4
- DATE 3-14
 - initial values 3-33
- DATE_AND_TIME 3-14, 3-33
- day 3-15
- deceleration ramp 6-17
- decimal 3-11, 3-33
- declarations
 - array 3-34
 - array of structures 3-40
 - array of structures w/arrays 3-44
 - blown fuse 3-31
 - Bottom table 3-24
 - hardware 3-20
 - Init. Val. field 3-44
 - I/O Pt. field 3-30
 - keystrokes 3-24
 - Long name field 3-33
 - menu 3-20, 3-24
 - Name field 3-28
 - software 3-23
 - structures 3-37
 - tables 3-23, 3-24
 - Top table 3-24
 - Type field 3-29
- default values, setup data 6-13
- define axes, number of 6-11
- delete
 - column 3-19
 - element 2-15
 - network 3-45
 - row 3-19
- delete command 3-25
- delete file, RAMDISK 4-23
- dependency list 4-18
- derivative
 - control 5-4
 - gain 6-19
- diagnostic error codes I-1
- diagnostics *see troubleshooting*
- dialog box, define 1-12
- digitizing axes 5-1, 6-7, 6-13
- DINT double int 3-12
- disk operations
 - flash 4-25
 - RAM 4-23
- display status Processor menu 4-29

documentation, comments 2-3, 3-48
dollar sign 3-13
DONE 3-7
DOS, installing software 1-6
Download, Module menu
 Monitor options 4-5
 procedure 4-2
 troubleshooting 4-3
DWORD double word 3-11

E

edit field command 3-25
edit network 3-45
elements
 array 3-34
 ladder logic 3-7
 network 2-1
EN 3-6, 3-7, 3-9
Enable
 Force table 4-10
 View Variables table 4-9
encoder 6-7
 driver 6-7
 type 6-7
end of module 2-12
Energize coil 3-3
error codes, diagnostic I-1
errors
 accessing C-1
 C-stop C-1
 E-stop C-1
 PiCPro functions B-1
 programming C-1
 timing C-1
E-stop
 define C-1
 errors
 bit locations C-4
 hex value C-4
Evaluate functions 3-4
extended/expanded memory H-1
extensible functions 3-8
external 3-26
E_ERRORS function C-2

F

fast inputs
 software declarations 3-32
FAST_QUEUE function, programming
 error C-1

faults, diagnostic I-1
feed forward
 compensation network 5-4
 percent 6-19
feedback
 analog input 6-9, 6-12
 encoder 6-8
 resolver 6-8
 TTL 6-10
feedback device
 analog input 6-7
 encoder 6-7
 inductosyn 6-7
 polarity 6-19
 resolver 6-7
 stepper 11-1
 TTL input 6-7
feedback units, define 6-1, 6-15
file
 extensions *see Appendix A*
 names 2-11, A-3
 transfer 4-23, 4-25
Filter functions 3-4
find
 coils 3-50
 function block 3-50
 network 3-47
 Prev. 3-52
 Replace 3-51
 variables 3-50
find command 3-27
Flash disk 4-18
 Processor menu 4-25
FMSDISK
 list files 4-25
 retrieve file 4-25
 update 4-25
following error
 define 5-4
 limit 6-20
Force 4-10
form feed 3-13
formula, ending ratio 7-10
function block
 add to network 2-15
 declare 3-29
 define 2-4, 3-4
 find 3-51
 raplace 3-51
 Type 3-29
functions

- add to network 2-15
- CD 3-7
- chain 3-9
- CU 3-7
- data flow 3-9
- define 2-3, 3-4
- DONE 3-7
- EN 3-6, 3-7
- execute 3-4, 3-6
- extensible 3-8
- IN 3-6, 3-8
- inputs 3-4, 3-6, 6-4
 - data types D-1
- number of inputs 3-8
- OK 3-6, 3-7
- OUT 3-6
- outputs 3-4, 3-6, 6-4
 - data types D-1
- Q 3-7
- REQ 3-7

fuse blown 3-31

G

- global 3-26
- GLOS, define 5-2
- Group 4-10

H

- hardware
 - declarations 2-5, 3-20
 - declarations table 2-5
- headings, in ladder printout 4-13
- help
 - Workstation menu 1-13
 - <F9> key 1-13
- Hex-86 dump 4-18
- hexadecimal
 - convert 3-11, 3-32
 - numbers 3-11
- horizontal
 - menu 3-19
 - wire 3-2
- hour 3-15

I

- IDN 10-5
- IN 3-10
- in position flag 6-20
- IND 4-7
- index number 3-36

- inductosyn 6-7
- INF 4-7
- Initial Value field 2-4, 3-44
- input
 - boolean 3-31
 - Data In 3-10
 - location 3-31
 - number of 3-8
 - physical 3-20, 3-31
 - to functions 3-9, 3-10
- insert
 - column 3-19
 - network 3-45
 - row 3-19
 - table entry 3-28
- insert command 3-25
- installing software, abort option 1-6
- INT integer 3-12

integral

- control 5-4
- error limit
 - minus 6-20
 - plus 6-20
- gain 6-19

interrupt number H-1

iteration

- command position 5-4
- command segments 5-4
- rate 6-20
- servo calculations 5-4

iterator data

- define 6-1
- descriptions 6-16

I/O function blocks 3-4, 4-23

I/O Pt. field 2-5, 3-30

J

- Jump 2-3, 3-48
- Jump to label 3-17
- Jump to subroutine 3-17

K

keys

- <Alt> 1-13
- 1-13
- <Enter>, accept 1-13
- <Esc>, cancel 1-13
- <F10>, accept 1-13
- <F8>, list 1-13
- <F9>, help 1-13

<Ins>, arrow 1-13

L

Label 2-3, 3-48

ladder units, define 6-1, 6-15

Latch coil 3-3

LED error codes I-1

library size, UDFB 8-5

line feed 3-13

LINT long int 3-12

list files

 FMSDISK 4-25

 RAMDISK 4-23

Long Name field 2-4, 3-33

longname 3-19

LREAL long real 3-12

LWORD long word 3-11

M

machine I/O, PiC system 5-2

main menu 2-7

mark variables 3-27

mark, network 3-38

master data telegram (MDT) 10-3

master synchronization telegram (MST) 10-3

master/slave, application 7-1

math coprocessor I-1

MDT

 structure 10-4

member 3-37

memory

 extended/expanded H-1

memory, PiC900 4-2

menu

 define 1-14

 label 2-7

 PiCPro 2-7

message, on-screen 1-14

millisecond 3-15

minute 3-15

modify command 3-25

module

 animate 2-8

 close 2-17

 components 2-1

 create 2-10

 end of 2-12

 hardware 3-20

 name 2-11

 new 2-12, 2-16

 open 2-11, 2-16

 Prev Function view 3-52

 print 2-8

 save 2-17

 save As 2-17

 User Function view 3-52

 zoom view 3-52

module menu 2-8

modules, PiC900 3-20

moniter application program 4-5

month 3-14

motion control

 evaluate axis motion 5-6

 introduction 5-1

 perform move 5-6

 PiC 5-2

 prepare axis 5-6

 start move 5-6

motion functions 3-4

move, network 3-38

MS-DOS, user interrupt H-1

N

Name field 3-31

NAN 4-7

negative transition (n.c.) 3-3

negative transition (n.o.) 3-3

NEMA SERCOS Specification 10-1

nesting, UDFB 8-5

network 2-1

 add element 2-15

 copy 3-48

 create 2-13, 2-14, 2-15

 cut and paste 2-16

 delete 3-45

 delete element 2-14

 design 2-18

 edit 2-13, 3-45

 elements 2-2, 2-3

 escape 2-16

 example 2-2

 execute 2-2

 Find 3-47, 3-48

 insert 3-45

 Jump 3-48

 Label 2-3, 2-13, 3-48

 mark 3-47

 move 3-48

 number 2-2, 2-13

 on-line edit 3-45

 open 2-14

- save 2-16
 - size 2-3
 - unmark 3-48
 - network menu 3-1
 - new line 3-13
 - Next command 3-49
 - next command 3-27
 - node
 - connect 4-28
 - set 4-28
 - node number 6-5
 - non-boolean i/o 3-7
 - normally Closed 3-3
 - normally Open 3-3
 - Numeric
 - constants 3-12
 - convert 3-32
 - data types 3-12
- O**
- octal 3-11, 3-32
 - OK 3-6, 3-9
 - on-line edit, network 3-45
 - open loop, define 5-1
 - operator interface device, PiC system 5-2
 - output
 - Data Out 3-10
 - of functions 3-6, 3-9, 3-10
 - physical 3-30
- P**
- patching 3-46
 - peer-to-peer, PiCPro 4-27
 - personal computer, purpose 5-2
 - phases
 - SERCOS 10-7
 - PiC Profile
 - data flow 7-5
 - define 7-1
 - define units 7-8
 - graph 7-10
 - other features 7-16
 - PiC900 system
 - Port1 *see workstation* 1-5
 - start from workstation 4-22
 - PiCPro
 - define 1-3, 5-2
 - errors B-1
 - menus 2-7
 - PiCServoPro
 - define 1-3, 5-2
 - development tools *see PiCPro, Servo Setup, and PiC Profile*
 - start-up 1-11
 - PiCTune *see tuning*
 - screen 6-26, C-3
 - view errors C-3
 - PiC/SERCOS network 10-1
 - PID loop *see proportional, integral, derivative*
 - define 5-4
 - setup 6-19
 - pop-up, define 1-12
 - position loop
 - control 5-3
 - PiC 5-2, 5-5
 - position loop data
 - define 6-1
 - descriptions 6-19
 - positive transition (n.c.) 3-3
 - positive transition (n.o.) 3-3
 - power reset, Processor menu 4-27
 - Prev Function 3-52
 - Previous command 3-49
 - print
 - declarations 4-13
 - ladder format 4-13
 - longnames 4-14
 - module 2-13
 - .LST file 4-13
 - printer
 - extended ASCII 1-17
 - troubleshooting 4-15
 - Workstation menu 1-16
 - printout characteristics 4-14
 - pro file
 - change name 7-17
 - define 7-5
 - Processor menu 4-21
 - profile
 - example 7-15
 - repeat 7-12
 - with RATIOPRO 7-16
 - profile data
 - create new file 7-7
 - example 7-10
 - initialize 7-15
 - make a function 7-14
 - open existing file 7-7
 - print 7-18
 - save 7-14
 - segments 7-9

- profile (ratio), example 7-4
- programming errors
 - bit locations C-6
 - define B-1, C-1
 - hex value C-6
- project manager
 - description 1-18
 - using 1-19
- proportional
 - control 5-4
 - gain 6-19
- purge unused variables 3-27
- P_ERRORS function C-2

Q

- Q 3-7
- queue
 - active 5-4
 - define 5-4
 - next 5-4
 - number 5-4

R

- RAMDISK 4-23
 - delete file 4-24
 - list files 04 - , 4-23
 - Processor menu 4-23
 - rename file 4-24
 - retrieve file 4-24
 - send file 4-24
- ratio profile, background 7-1
- RATIOPRO function
 - programming errors C-6
- real time display 4-5
 - Animate 4-5
 - View Variable List 4-8
- real time interrupt, iteration 5-4
- recursion 8-5
- reference card D-1
- reference, ignore limits 6-18
- rename file, RAMDISK 4-23
- replace 3-51
- Replace command 3-51
- REQ 3-6
- requirements 1-4
- resolver 6-7
- restart 4-27
- restore, Processor menu, define 4-26
- retentive variables 3-23
 - warm restart 4-22

- retrieve file
 - FMSDISK 4-25
 - RAMDISK 4-23
- return from subroutine 3-17
- rollover
 - on position 6-17
 - position 6-17

S

- save
 - module 2-16
 - .BIN 4-26
- scaling data
 - define 6-1
 - descriptions 6-15
 - example 6-15, 6-16
 - input 6-15
 - output 6-15, 6-16
- scan
 - count, real time display 4-5
 - ladder scan 4-21
 - operating system overhead 4-21
 - start 4-22
 - stop 4-22
- screen display lines 1-14
- second 3-15
- semaphore flag 9-10
- send file, RAMDISK 4-23
- SERCOS
 - and standard motion.lib functions 10-10
 - and variables 10-10
 - application notes 10-17
 - AT 10-3
 - cyclic data 10-9
 - functionfunction blocks 10-2
 - introduction 10-1
 - MDT 10-3
 - MST 10-3
 - network (PiC) 10-1
 - phases 10-7
 - scaling units 6-15
 - service channel data 10-9
 - setup 10-12
 - specification (NEMA) 10-1
 - telegrams 10-3
- service channel data
 - SERCOS 10-9
- servo 5-1, 6-5, 6-11
 - axes calculations procedure 5-3
 - library, create 6-23
 - programming overview 5-6

- system define 5-1
- Servo setup
 - data flow 6-3
 - define 5-2, 6-1
- servo upgrade (SUG), rate 6-20
- Set coil 3-3
- set network node id, Processor menu 4-29
- setup
 - SERCOS 10-12
- setup data 5-1
 - and motion control 5-1
 - create a new file 6-5
 - enter 6-4
 - handled by PiC 5-8
 - initialize 6-24
 - iterator data 5-1
 - list of 6-14
 - make a function 6-23
 - position loop data 5-6
 - print 6-31
 - processor menu 6-5
 - scaling data 5-6
 - type of axes 6-6
- single quote 3-13
- slot, hardware 3-21
- soft bit memory H-1
- software
 - declarations 2-4, 3-10, 3-23
 - declarations table 2-4
- software declarations command
 - delete 3-25
 - edit field 3-25
 - find 3-27
 - insert 3-25
 - mark 3-27
 - modify 3-25
 - next 3-27
 - purge unused variables 3-27
- software limit
 - lower 6-18
 - upper 6-18
- srv file
 - define 6-6
 - save 6-23
- status of control 4-29
- step reference card F-1
- stepper
 - and standard motion.lib functions 11-2
 - and variables 11-2
 - axis module 11-1
 - configure 11-2
 - software 11-1
- String
 - Appl Note N-3
 - data type 3-13
 - declare 3-29
 - display value 3-32
 - functions 3-4
 - initial values 3-32
 - outputs 3-15
- structures
 - arrays of 3-40
 - declare 3-37
 - define 3-37
 - members 3-37
 - size 3-37
 - with arrays 3-38
- switches, command line H-1

T

- tab 3-13
- task 3-4, 4-16
 - and I/O functions 9-4
 - and motion functions 9-4
 - creating 9-3
 - defintion 9-1
 - interlocking data 9-10
 - interruption 9-6
 - priority 9-6
 - programming 9-2
 - rules 9-3
 - UDFB comparision 9-1
- telegrams
 - SERCOS 10-3
- TIME duration 3-15
 - initial values 3-33
- Time of day 3-15
- time read/set, Processor menu 4-26
- Timer function blocks 3-4
- time, axes G-1
- TIME_OF_DAY 3-15, 3-33
- timing error, define C-2
- TME_ERR? function C-2
- troubleshooting
 - Animate 4-5
 - communications error 4-8
 - download 4-3
 - PiC900 system 4-30
 - printer 4-15
- TTL
 - binary encoding 6-10
 - feedback 6-10

- gray code 6-10
- input 6-7
- tuning
 - data flow 6-25
 - define 6-1
 - download changes 6-27
 - inputs, marking 6-27
 - introduction 6-25
 - PiCTune screen 6-27
 - save changes 6-27
 - view outputs 6-28
- Type field 2-4
- U**
- UDFB 4-16
 - attributes 3-10
 - change 8-13
 - create 8-10
 - create module 8-2
 - definition 8-1
 - errors 8-10
 - guidelines 8-1
 - handle data 8-2
 - inputs/outputs 8-4
 - library size 8-5
 - module example 8-6
 - nesting 8-5
 - recursion 8-5
 - software declarations 8-8
 - store 8-12
 - template 8-1
 - type check 8-12
 - use 8-12
 - variables
 - IN 8-9
 - marking 8-9
 - OUT 8-9
 - view Prev function 3-52
 - view user function 3-52, 8-12
- UDINT unsigned double int 3-12
- UINT unsigned int 3-12
- ULINT unsigned long int 3-12
- units
 - axis 6-15
 - feedback 6-15
 - ladder 6-15
- Unlatch coil 3-3
- unmark network 3-48
- update FMSDISK 4-25
- update rate 6-20
- update UDFBs H-1

- user defined function blocks 3-4
- User Function 3-52

V

- variables 3-10
 - attributes 3-28
 - declare 3-27
 - default values 3-27
 - define 2-3
 - external 3-26
 - find 3-49
 - force 4-10
 - global 3-26
 - group 3-37
 - IN 3-26
 - initial values 3-32
 - Name 3-27
 - OUT 3-26
 - replace 3-51
 - retentive 3-26
 - types 3-10
 - unique 3-15
- velocity
 - filter 6-17
 - fast 6-17
 - slow 6-17
 - limit 6-17
 - profile 7-1
- velocity loop
 - define 5-6
- vertical
 - menu 3-19
 - wire 3-2
- View
 - errors 6-28
 - loop status 6-28
 - on-line servo variables 6-28
 - Prev. Function 3-52
 - setup parameters 6-28
 - User Function 3-52
 - zoom 3-52
- View Variables, Monitor menu
 - real time display 4-8
- W**
- wires
 - both 3-2
 - define 2-3
 - horizontal 3-2
 - toggling 3-2
 - vertical 3-2

WORD 3-11

workstation 1-4

 cable to PiC900 1-5

 configure 1-5, 1-14

 Workstation menu 1-14

X

Xclock functions 3-4

Y

year 3-14

Z

Zoom 3-52

Symbols

.LST file 4-13