ACCU ELECTRIC MOTORS INC

USA: (888) 932-9183 CANADA: (905) 829-2505

- Over 100 years cumulative experience
- 24 hour rush turnaround / technical support service
- Established in 1993

The leading independent repairer of servo motors and drives in North America.

Visit us on the web:

www.servo-repair.com

www.servorepair.ca

www.ferrocontrol.com
www.sandvikrepair.com
www.accuelectric.com

Scroll down to view your document!

For 24/7 repair services :

USA: 1 (888) 932 - 9183
Canada: 1 (905) 829 -2505

Emergency After hours: 1 (416) 624 0386

Servicing USA and Canada

MOTIONTECHNOLOGYDIVISION

110 Fordham Road
Wilmington, MA 01887
(508) 988-9800
Fax (508) 988-9940

Part# 903-075212-80
List Price: $30 U.S.
February, 1996
Rev E

S C 7 5 0 S E R I E S

Programmable, Digital Brushless Servocontroller

ServoBASIC *Plus*™ Programming Manual

Version 2.8

Table of Contents

1 Overview of the SC750 Series

In this chapter This chapter introduces the SC750 series servocontroller programming environment. Topics covered are:

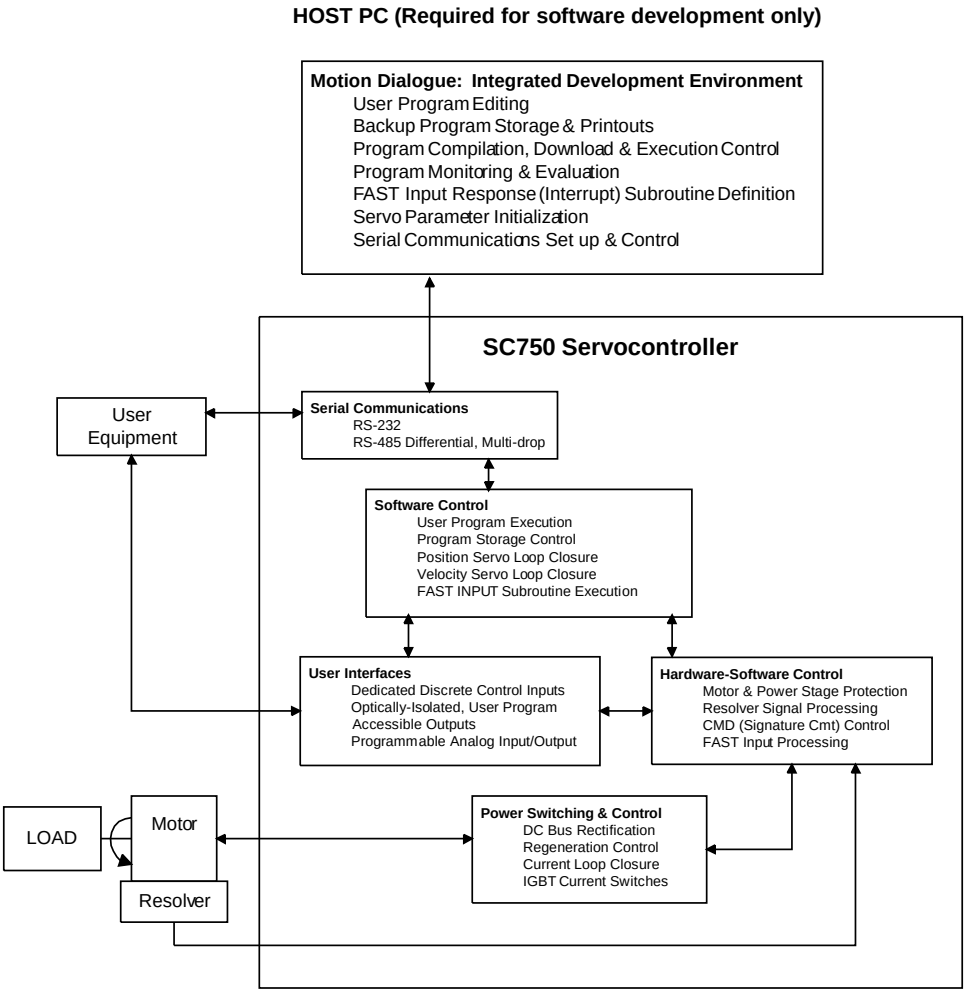
- Functional block diagram
- How to use this manual

1.1 Functional Block Diagram

The functional block diagram of the SC750 Servocontroller and related equipment is shown on the following page. The primary elements of the diagram are:

- Host PC
  - Initial development of ServoBASIC *Plus*™ software for the SC750.
- SC750 Servocontroller
  - Embedded controller performing: software execution; digital servo control algorithms; diagnostics; serial and discrete I/O interfaces and power switching control.
- User Equipment
  - Process control logic, sensors, display and control panels, and related equipment required to incorporate the SC750 into user applications.
- Load, Motor/Resolver
  - Power Drive Output/Feedback Sensor for the SC750.

Functional diagram



1.2 How to Use This Manual

This manual is intended to provide a detailed description of the SC750 ServoBASIC *Plus*™ programming language and development environment.

Documentation of the SC750 Series servocontrollers is provided in three separate manuals:

- I - SC750 Manual: Installation and Hardware Reference
- II - SC750 Manual: ServoBASIC *Plus*– Programming
- III - SC750 Manual: ServoBASIC *Plus*– Reference

2 Setting Up for Programming

In this chapter

This chapter explains how to set up your SC750 Servocontroller for programming. Topics covered are:

- Preliminary requirements
- Host PC requirements
- Setting up communications

2.1 Preliminary Requirements

Introduction

This section covers the preliminary steps that must be performed before programming can begin.

Equipment preliminaries

Upon completion of the installation instructions from the "SC750 Installation and Hardware Reference Manual," the servocontroller is:

- Connected to the controlling computer
- Connected to external I/O devices, the motor, and power
- Addressed if the unit is to be operated under RS-422 or RS-485 from a single computer with other servocontrollers

The servocontroller is ready for the following steps if AC power is applied to the unit and the seven segment display on the front panel of the unit indicates a fault display code of zero or eight.

Note: *In addition, the amber BUS Active LED on the SC752/SC753 units should be lit.*

If either of these conditions are not met, refer to Section 4.2 of the "SC750 Installation and Hardware Reference Manual."

BASIC programming language knowledge

Effective use of the SC750 drive requires a working knowledge of BASIC programming. Although an overview of general BASIC programming is given, you should study a guide for BASIC programming, if needed.

2.2 Host PC Requirements

To program the SC750, an IBM PC or compatible computer is required. Only after the SC750 is programmed can a dumb terminal be used to provide serial communications to the SC750.

Motion Dialogue

Motion Dialogue requires a minimum of the following:

- DOS version 3.3 or later
- A 5 1/4 inch 360 KB or a 3 1/2 inch 720 KB disk drive
- One serial COM port (Two if a serial port (optional) mouse is used)
- A minimum of 450 KB memory at the DOS prompt (512 or more for optimum performance)

The PC will serve as the host for initial set up and software development for the SC750. Supplied with the SC750 is a software disk containing the Motion Dialogue programming environment.

Motion Dialogue is invoked at the DOS command prompt by inserting the supplied disk in the A drive of your PC and entering A:750↵.

2.2.1 New Power On Display Message

When AC control power is applied to the SC750 controller the red seven segment LED status display will flash the letters

P A C S C I - X Y ,

where X Y is the axis address of the control, displayed in hexadecimal format (Eg. Axis 255 = FF, Axis 1 = 01, Axis 14 = 0E).

After this power up sequence the status display will function with the system status display codes indicated in the controller reference documentation.

2.3 Setting Up Communications

Introduction

This section covers the hardware connections required to interface a Host PC to an SC750 series servocontroller.

Serial port requirements

The requirements for the terminal are:

- RS-232, RS-485, or RS-422 serial communication
- 9600 baud transmission rate

Refer to the “SC750 Installation and Hardware Reference Manual” for additional detail on hardware connections.

3 ServoBASIC *Plus* Programming Environment

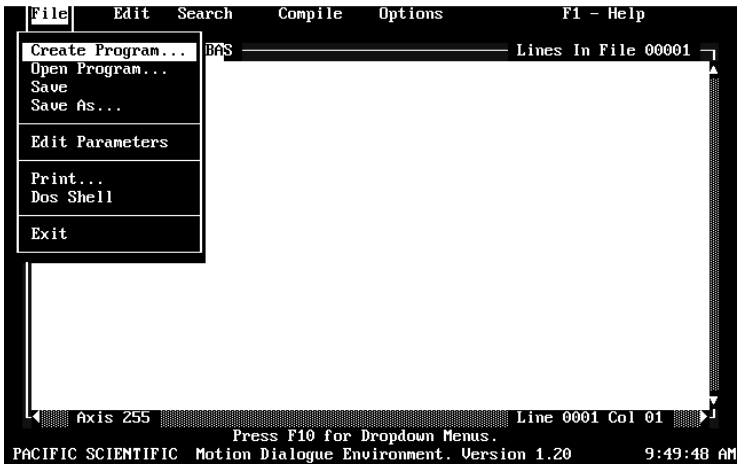
In this Chapter

This chapter describes Motion Dialogue: The ServoBASIC *Plus* programming environment. Topics covered include:

- Introduction to the Motion Dialogue environment
- Options menu - Environment set up
- Edit menu
- Search menu
- File menu
- Help menu

3.1 Introduction to the Motion Dialogue Environment

Motion Dialogue is an interactive program development utility. To invoke Motion Dialogue place the supplied diskette in the A: drive of your PC and type 750 ↵. This will place the user in the editor screen mode of Motion Dialogue (shown below).



Menu

The top level command menu of Motion Dialogue is accessed via pull down menus obtained by pressing the **F10 key** on the PC keyboard. Exiting from the menu windows is performed by pressing the **ESC key**.

Menu functions

The top level menu supports the following editing functions:

- File (Utilities)
- Edit/Search Utilities
- Options (Environment Set Up)
- Help

Additional top level menu functions include:

- Servo Set Up
- Compilation/Run/Download

Servo Set Up and Compilation features are discussed in detail in sections 6 and 7 of this manual.

Function key table

The table below lists Motion Dialogue's special function keys.

Menu	Function	Accelerator Key
File		Alt F
	Create Program	
	Open Program	
	Save	
	Save As	
	Edit Parameters	
	Print	
	Dos Shell	
	Exit	

Menu	Function	Accelerator Key
Edit		Alt E
	Cut	Shift + Del
	C**o**py	Ctrl + Ins
	Paste	Shift + Ins
	Clear	Del
Search		Alt S
	Find	
	Repeat last find	
	Change	
Compile		Alt C
	Compile Program	
	Run Program	
	Rese**t** Program	
	Co**n**tinue Program	
	Stop Program	
	Output Window	
	Set **B**reakpoint	
	Cl**e**ar All BRkpts	
	Variables ...	
	Real Time **F**unction Call	
Options		Alt O
	Display	

Menu	Function	Accelerator Key
Options (cont.)	**P**ort Configuration	
	Controller Set Up	
	Unconfigure	
	Axis Selection	
	Download Source File	
	Download BIN File	
	Upload Source File	
Help		Alt H
	Index	
	General	
	Editor Commands	
	About	

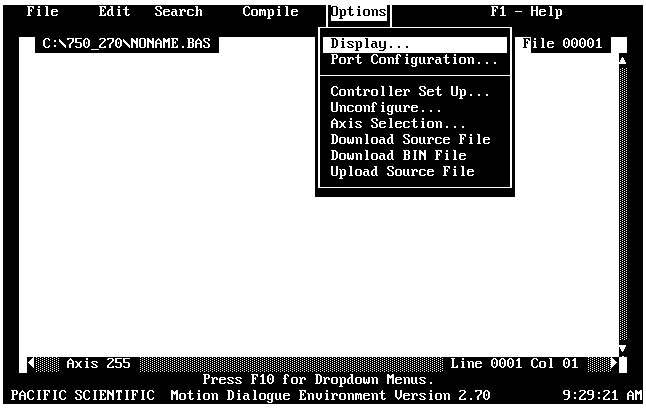
Note: *Accelerator keys allow you to access a function without using the pull down menu within the editor window.*

3.2 Options Menu - Environment Set Up

Introduction

The Options Menu permits configuration of the video monitor, the computer's serial communication port and servocontroller set up. Servocontroller set up is described in section 6 of this manual.

Options screen



Procedure

To select the “Options” menu, perform the following procedure:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “Options.”

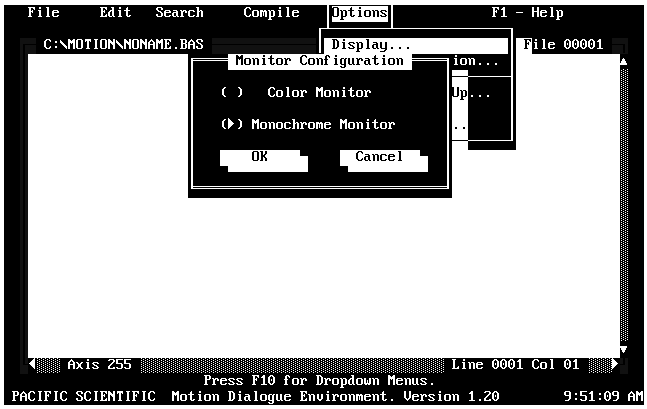
The Options menu is now displayed on your screen.

3.2.1 Display Set Up

Introduction

Monitor display set up allows for either monochrome or color display.

Display screen



Procedure

To select “Display”, perform the following procedure:

1. Using the arrow keys (↑ ,↓), move the cursor to highlight “Display” and press ↵ . The Monitor Configuration Screen will now appear.
2. Now, using the arrow keys (↑ ,↓), select the appropriate screen configuration “Color Monitor” or “Monochrome Monitor.”.
3. <Tab> to OK and press ↵ .

Note: *Set up selections will only be saved to a disk configuration file by pressing the enter key after making your selections.*

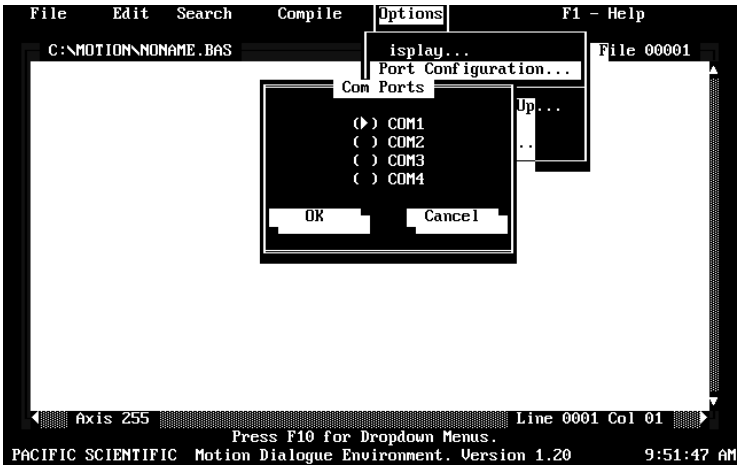
3.2.2 Communications Set Up

Introduction

The serial communications set up selects the COM port on the PC which is used by Motion Dialogue to communicate with the servocontroller. This provides the ability to check serial I/O programmed by the user.

Note: *The Host PC will always utilize 9600 baud, one stop bit, one start bit and no parity. This is what the SC750 communications format expects and how Motion Dialogue configures the PC’s serial port.*

Port configuration screen



Procedure

To select “Port Configuration”, perform the following:

1. Using the arrow keys (↑ , ↓), move the cursor to highlight “Port Configuration” and press ↵ . The Com Ports screen will now appear.
2. Again, using the arrow keys (↑ , ↓), select the appropriate port configuration (COM1, COM2, COM3, or COM4).
3. <Tab> to OK and press ↵ .

Note: *Set up selections will only be saved by pressing the enter key after making your selections.*

3.2.3 Axis Selection

This function permits selection of the multidrop subsystem address for the SC750 servocontroller, if multidrop is used.

Note: *The default axis when invoking Motion Dialogue is 255. This axis corresponds to RS232 communications.*

Multidrop communications is supported on axis 1 through 254. Up to 32 controllers can be addressed using multidrop. Each controller must have a unique address, selected by the S1 dip switch. Refer to Section 3.1.1 of the “SC750 Installation and Hardware Reference Manual” for additional information on setting the controller multidrop address and cabling requirements for multidrop installations.

3.2.4 Download of Pre-Compiled (.BIN) Programs

Motion Dialogue provides the ability to download a compiled ServoBASIC Plus program. Compiled SC750 programs have a .BIN file extension

This feature, performed out of the Motion Dialogue options window, will permit downloading (and running) programs, without invoking compilation of a .BAS source program. Hence, the .BAS source file is not required to perform this function.

3.2.5 Program Source Code Storage on the Controller

This section describes utilities that permit the archiving and retrieval of ServoBASIC *Plus* source code within the controller's nonvolatile memory.

3.2.5.1 Downloading Program Source Code (.BAS) to the Controller

The capability to download program source code for storage in the controller non-volatile memory is now a ServoBASIC *Plus* feature. This function is performed within the Motion Dialogue Option Window.

Storage of the source program is limited by the amount of non-volatile memory available on the controller. If you want to put the source program on the controller, then there must be enough memory on the controller for the executable file and the source program. The source program will require the same number of memory bytes on the controller as it requires on your disk.

Status bars illustrate the progress of downloading source code to the controller.

3.2.5.2 Uploading Program Source Code (.BAS) to Motion Dialogue

The Source program upload function is performed via the options window in Motion Dialogue. This function provides the ability to extract a ServoBASIC *Plus* program saved in non-volatile memory and display it in the edit screen.

Status bars illustrate the progress of uploading source code from the controller.

3.3 Edit Menu

The Edit Menu allows you to perform the following operations:

- Typing text/cursor movement
- Cutting/pasting text
- Clearing text buffer

Note: The Edit menu only allows access to those functions available for the edit you are performing. For example, if a block of text has just been highlighted it can be cut, copied, or cleared BUT cannot be pasted. Therefore, the paste option is not highlighted.

3.3.1 Typing Text and Cursor Movement

Introduction

Typing text into the Motion Dialogue editor is performed while in the editor screen mode. The editor screen is accessed from the top level command menu by pressing the ESC key after highlighting the “Edit” selection.

Note: The maximum line length supported by the editor is 255 characters. The Edit screen scrolls sideways after the cursor moves beyond 80 characters.

3.3.2 Cutting and Pasting Text

Introduction

Cutting and pasting text is performed while in the Edit menu mode.

Edit screen

Procedure

To "cut text," perform the following procedure:

1. Press Shift plus the arrow key (↑ , ↓) to highlight the portion of text to be cut.
2. Press the F10 key to access the pull down menu.
3. Using the arrow keys (← , →), move the cursor to highlight "Edit."
4. Now, using the arrow keys (↑ , ↓) highlight "Cut" and press ↵. (The text is now stored in the Clipboard.)

Note: *Text can also be cut by pressing Shift plus Del after highlighting the designated text.*

To "paste text", perform the following:

1. While in the Edit mode, move the cursor to the portion of your program in which text is to be placed.
2. Press the F10 key to access the Edit menu.
4. Now, using the arrow keys (ì , Å) highlight "Paste" and press ↵. (Text stored in the Clipboard will be placed in this portion of the program.)

Note: *Text can also be pasted by pressing Shift plus Ins after selecting the portion of the program for text location.*

3.3.3 Copying Text

Introduction

Copying text is also performed while in the Edit menu mode.

Procedure

To "copy text", perform the following procedure:

1. Press Shift plus the arrow key (↑ , ↓) to highlight the portion of text to be copied.
2. Press the F10 key to access the pull down menu.
3. Using the arrow keys (← , →), move the cursor to highlight "Edit."

4. Now, using the arrow keys (↑ , ↓) highlight "Copy" and press ↵. (The text is now being stored in the Clipboard.)
5. While in the Edit mode, move the cursor to the portion of your program in which text is to be placed.
6. Press the F10 key to access the Edit menu.
7. Now, using the arrow keys (↑ , ↓) highlight "Paste" and press ↵ . (Text stored in the Clipboard will be placed in this portion of the program.)

Note: *Text can also be copied by pressing Ctrl plus Ins after highlighting the designated text, moving to the new text location and pressing Shift plus Ins.*

3.3.4 Clearing Text

To remove text, perform the following procedure.

Procedure

1. Move your cursor to the location of text to be removed. Press Shift plus the arrow key (↑ , ↓) to highlight the portion of text to be deleted.
2. Press the F10 key to access the main menu.
3. Using the arrow keys (← , →), move the cursor to highlight "Edit."
4. Now, using the arrow keys (↑ , ↓) highlight "Clear" and press ↵ . (The text is now removed from the program.)

Note: *Text can also be removed by pressing del after highlighting the designated text.*

Edit menu functions

The table below lists accelerator keys for the Edit functions.

Function	Accelerator Key
Cut	Shift + Del
Copy	Ctrl + Ins
Paste	Shift + Ins
Clear	Del

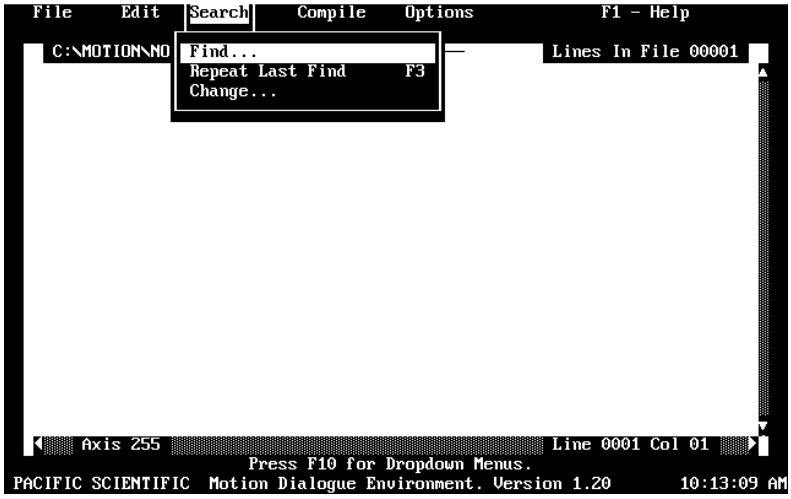
3.4 Search Menu

Introduction

The Search menu allows you to:

- Locate text within a program
- Change text within a program

Search screen



Procedure

To select the “Search” menu, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “Search.”
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the appropriate search function (Find, Repeat Last Find, or Change) and press ↵ .

Edit menu functions

The table below lists accelerator keys for “Search Menu” functions.

Function	Accelerator Key
Find	
Repeat Last Find	F3
Change	

3.4.1 Find Text

Introduction

Motion Dialogue allows you to locate a specific string of text.

Find screen

Procedure

To “find” a specific string of text, perform the following:

1. Using the arrow keys (↑ , ↓), move the cursor to highlight “Find” and press ↵ .
2. Enter the sequence you are searching for.
3. Using the <tab> or arrow keys (↑ , ↓), move the cursor to highlight OK and press ↵ .

Note: *Your cursor will move to the first position in your program where the sequence is located. To locate additional sequences either type F3 or return to the main menu (F10), cursor to “Repeat Last Find” and press ↵ .*

3.4.2 Repeat Last Find

Introduction

Motion Dialogue allows you to repeat the last “find query” made.

Procedure

To “Repeat last find,” perform the following procedure:

1. Using the arrow keys (↑ , ↓), move the cursor to highlight “Repeat Last Find” and press ↵ .
2. <Tab> to OK and press ↵ .

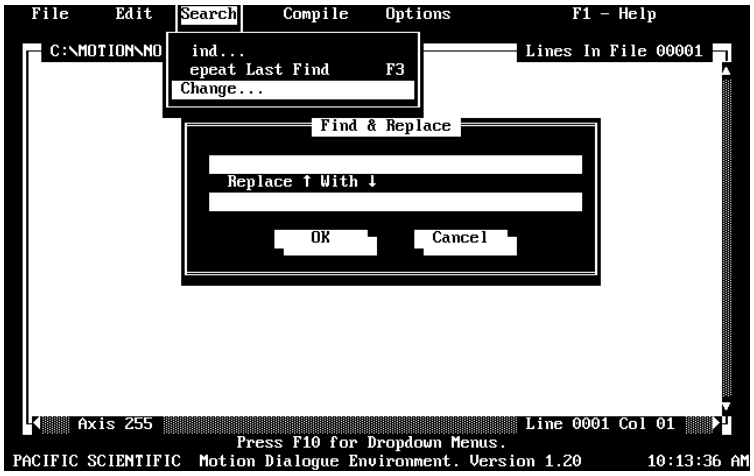
Note: *The accelerator key for the Repeat Last Find function is F3.*

3.4.3 Change

Introduction

Motion Dialogue allows you to replace existing text with new text.

Change screen



Procedure

To replace existing text with new text, perform the following:

1. Using the arrow keys (↑ , ↓), move the cursor to highlight "Change" and press ↵ .
2. Enter the text to be replaced.
3. Using the arrow keys (↑ , ↓), move to the next entry and enter the replacement text.
4. <Tab> to OK and press ↵.
5. Motion Dialogue prompts with the following:

   Change? (**Y**)es (**N**)o (**G**)lobal (**C**)ancel

   - Select Yes/No for individual replacement of changes.
   - Select Global to replace each occurrence of existing text with new replacement text.
   - Select Cancel to cancel the change operation.

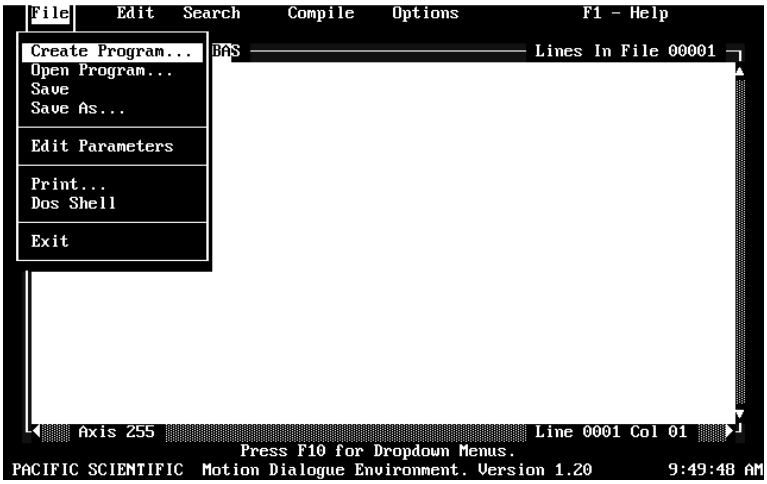
3.5 File Menu - Program Maintenance

Introduction

The File menu allows you to perform the following functions:

- Create program
- Open an existing program
- Save an existing program
- Save a new program
- Print a program
- Shelling to DOS
- Exit program

File menu screen



Procedure

To select the “File” option, perform the following procedure:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “File.”.
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the appropriate menu selection and press ↵ .

3.5.1 Creating a Program

To create a program:

- Begin typing in the screen editor environment

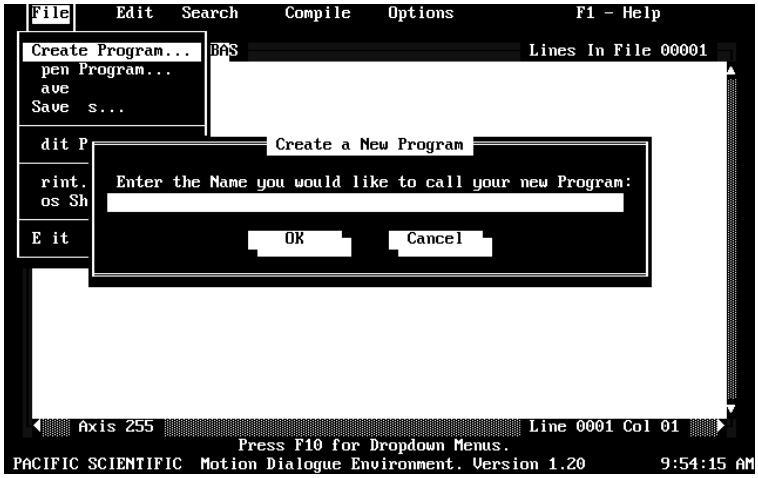
  or

- Select the “Create Program” selection from the File Menu.

Warning

New program automatically deletes the program present in the edit screen.

Create program screen



Procedure

To “create a program”, perform the following procedure:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “File.”
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the “Create Program” menu selection and press ↵ .

4. Motion Dialogue prompts you for the name of the program you are creating. Type in the name, <tab> to OK, and press ↵ .

5. You are now prompted for parameter set up information. There are certain parameters needed for any SC750 program. Using the arrow keys (↑ , ↓), select the option you prefer.

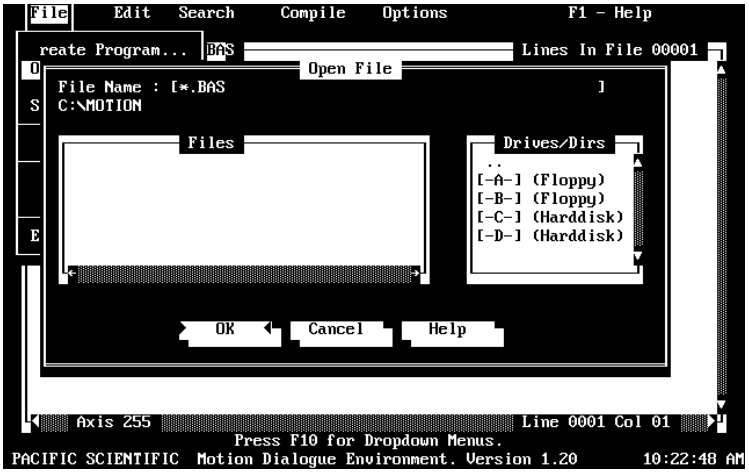
6. <Tab> to OK and press ↵.

Note: *If your controller has not been set up, Motion Dialogue will prompt you for controller set up information. Refer to Section 6.1, Controller Set Up for detailed information.*

3.5.2 Opening a Program

Motion Dialogue allows you to load existing programs into the screen editor.

Open program screen



Procedure

To "open" an existing file, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight "Open Program" and press ↵ .
4. Enter the name of the file to be loaded.

Note: *If you do not know the name of the file to be loaded, <tab> to the "Drives/Dirs" box and select the correct directory. A list of file names will appear in the "Files" box. <Tab> to the files box and using the arrow keys (↑ , ↓), move the cursor to highlight the desired file name and press ↵.*

5. Using the <tab> key move to OK and press ↵ .

3.5.3 Saving a Program

Introduction

Use Save Program to save the program currently in the edit screen as the filename specified in the filename window. Use the Save As selection to save the current program under a new filename.

Procedure

To save a program, perform the following procedure:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "Save" selection and press ↵ .

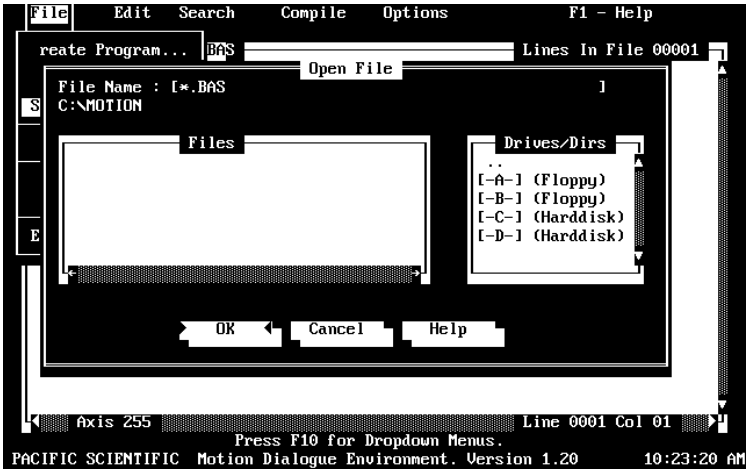
Note: *Only use the save command for programs that have already been named. For new programs you must use the "Save As" function.*

3.5.4 Save As Program

Introduction

When a new program has been created or you wish to save a program under a different name, the Save As function must be used.

Save As screen



Procedure

To save a new file, perform the following procedure:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "Save As" selection and press ↵ .
4. Enter the program name, <tab> to OK and press ↵ .

Note: *If you wish to save the program in a directory other than the default directory, <tab> to the "Drives/Dirs" box and select the correct directory.*

3.5.5 Printing a Program

To print a program that is currently in Motion Dialogue, perform the following procedure.

Procedure

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "Print" selection and press ↵ .

3.5.6 Shelling to DOS

To shell to DOS from Motion Dialogue, perform the following procedure.

Procedure

To access the DOS shell from Motion Dialogue:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "DOS Shell" menu selection and press ↵.
4. To return to the Motion Dialogue screen, type "exit" at the DOS prompt.

3.5.7 Exiting from Motion Dialogue

To exit from Motion Dialogue, perform the following procedure.

Procedure

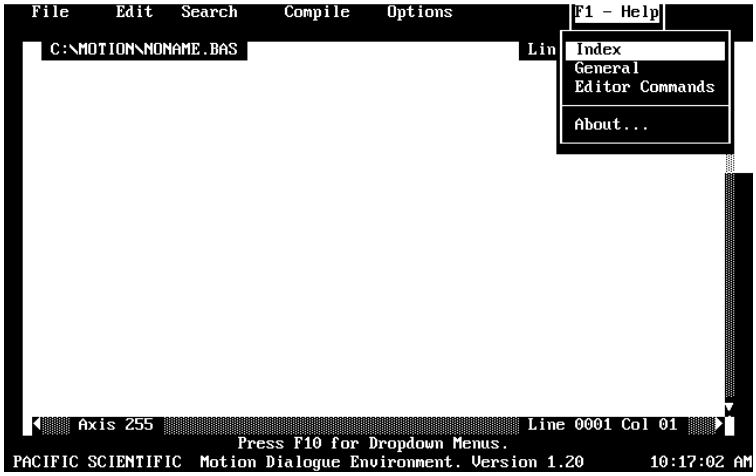
1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "File."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "Exit" menu selection and press ↵ .
4. Motion Dialogue asks you if you are sure you would like to exit. If yes, press ↵ .

3.6 Help Menu

Introduction

An on-line help screen can be accessed at any time within the Motion Dialogue program by either pressing the F1 key or by accessing the main menu (F10) and moving the cursor to Help.

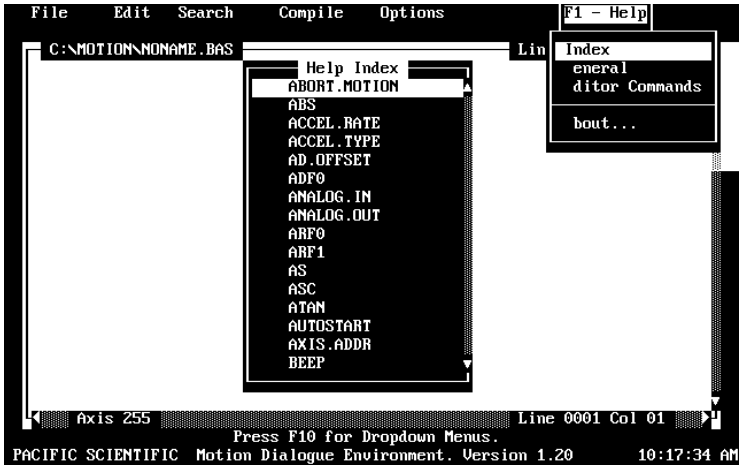
Help menu



3.6.1 Index of Commands

Motion Dialogue provides a list of ServoBASIC *Plus* commands. This list is contained within the help menu. The help screens for each command can be accessed via this list.

Index screen



Procedure

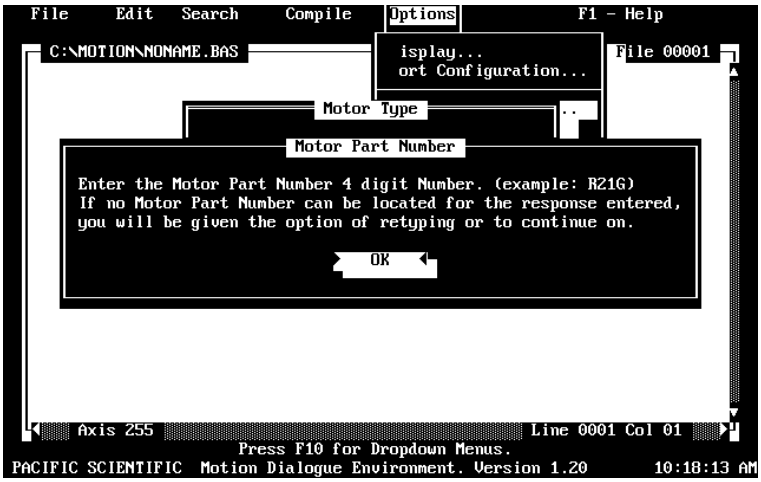
To access the index list, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Help."
3. Using the arrow keys (↑ , ↓), move the cursor to highlight the "Index" menu selection and press ↵ .
4. Using the arrow keys (↑ , ↓), select the command for which help is needed. Press ↵ .

   The help screen for the selected command will be displayed.

3.6.2 General Help

The Motion Dialogue on-line help is sensitive to the particular area of the screen in which you are working. For example, if you are working in the controller setup menu at the "Motor part number" prompt, pressing F1 will produce the following help screen:



3.6.3 Editor Commands

The editor help command provides a summary of editor "hot keys" and the associated definitions.

Editor help screen

Procedure

To access “editor help,” perform the following:

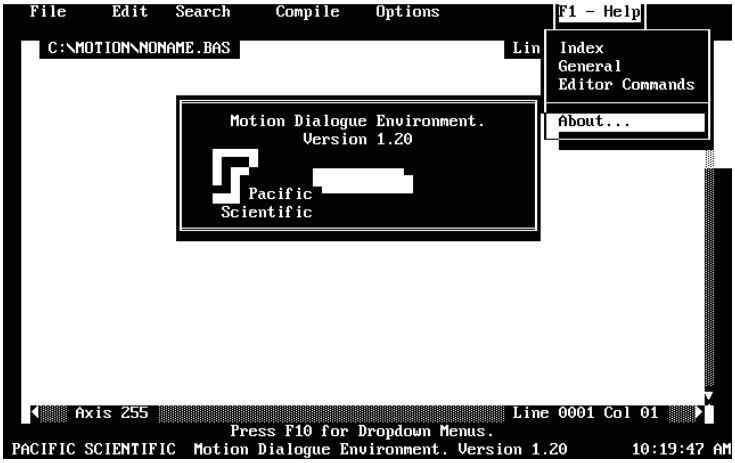
1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “Help.”
3. Using the arrow keys (↑ , ↓), move the cursor to highlight “Editor Commands” and press ↵ .

   The hot key list is now displayed.

3.6.4 About...

The “About...” option contains information pertaining to manufacturer, version number, etc.

About... screen



Procedure

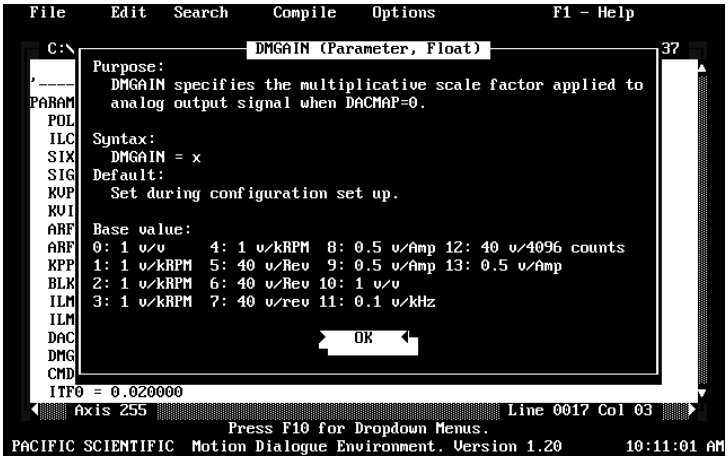
To access “About...” perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight “Help.”
3. Using the arrow keys (↑ ,↓), move the cursor to highlight the “About...” menu selection and press ↵ .

3.6.5 Help Within a Program

Motion Dialogue Help allows you to access information regarding specific commands while working within the program editor.

Help within program screen



Procedure

To access on-line help from within the editor, perform the following:

1. Move the cursor to the location of the command in question.
2. Press the F1 key. The associated help screen with appear.
3. Press the ESC key to return to the editor.

4 ServoBASIC *Plus* Language

In this chapter

This chapter describes the ServoBASIC *Plus* programming language. Topics covered are:

- ServoBASIC *Plus* program structure
- Understanding the ServoBASIC *Plus* instruction types
- ServoBASIC *Plus* functionality
- ServoBASIC *Plus* motion control
- Servo tuning and related instructions

4.1 ServoBASIC *Plus* Program Structure

The elements of a ServoBASIC *Plus* program consist of:

- Setup parameter definition
- User variable declaration
- Subroutines
- Interrupt subroutines

ServoBASIC *Plus*™

Program listing

Tabulated below is a program listing generated by Motion Dialogue prior to developing a specific motion control program. This "boiler-plate" is intended to assist in program development and serves as an aid in recognizing program sections. The individual program sections are now described in more detail in Section 4.1.1.

"Boiler-Plate Program Listing"

```
'————————Parameter Values Header————————
PARAMS START
POLECOUNT = 4
ILC = 13
SIX.THRESH = 430.664063
SIG.THRESH = 387.597656
KVP = 0.087086
KVI = 5.000000
ARF0 = 150.000000
```

```
ARF1 = 750.000000
KPP = 15.000000
BLKTYPE = 2
ILMT.PLUS = 100
ILMT.MINUS = 100
DACMAP = 1
DMGAIN = 1.000000
CMDGAIN = 1.000000
ITF0 = 0.020000
IT.THRESH = 60
KVFF = 0.000000
ADF0 = 1000.000000
AD.OFFSET = 0.000000
DMF0 = 1000.000000
COMMOFF = 0.000000
PARAMS END
‘————————Variable Definitions————————
‘ for help put cursor on -> DIM <- and
‘press the F1 key
‘————————Main Program————————
END ‘ Main Program
‘————————Subroutines————————
‘————————Interrupt Routines————————-
‘————————End of Program Listing————————
```

4.1.1 Program Sections

Setup parameter definition

This section of the program defines the power-on default parameters specifying servocontroller tuning and configuration. This section is preceded by the PARAMS START statement and terminated with the PARAMS END statement. Generation of this program section requires parameter upload from a servocontroller, by performing a Controller Setup, or by manually generating a setup parameter section following the format shown in the program listing.

User variable declaration

Motion Dialogue requires all user variable names to be defined using the DIM statement. This should be done prior to their use within a program. This program section must occur before any executable program statement.

User variables can be either:

- volatile (cleared when controller power is cycled)
- or non-volatile (retained when power if off)

All non-volatile user variables must be defined prior to the definition of volatile variables.

Main program section This portion of the program defines the overall motion control program. Following a power-on sequence, the instructions are executed sequentially from the first to the second, to the third, and so on. Subroutines, if used, can be called from the main program body. Interrupts, if used, must be enabled within the main program section. This section is terminated with a single END statement.

Subroutine definitions One or more subroutines that are called from within the main program are defined within this program section.

Interrupt routines Subroutines generated to support the interrupts are contained within this section of the program.

4.1.2 Line Format

One major difference between ServoBASIC *Plus* programs and programs produced with earlier versions of BASIC is that line numbers are not permitted. Program readability and structured appearance is further improved by indentation of text (enhanced by the memory feature of the tab key). This permits easier to read programming structures not permissible in interpreted BASIC, using line numbers.

Note: *Line lengths up to 255 characters wide are supported by Motion Dialogue.*

Example

```
DIM PEAK_VEL AS FLOAT     ' DECLARE USERVARS
IF VELOCITY > 1750 THEN         ' SET THRESHOLD
  IF VELOCITY > PEAK_VEL THEN         ' PEAK
DETECT
        PEAK_VEL = VELOCITY         ' SAVE PEAK
        ELSE
  ENDIF
ENDIF
```

Labels

Labels can be used to permit definition of instructions which alter the flow of program execution, such as GOTO and GOSUB. Using labels is discussed in more detail in section 4.1.5.

4.1.3 User-Defined Variable Names

Introduction

User-defined variables are *used with BASIC functions and statements for general programming tasks.* There are three basic types of user-defined variables:

- INTEGER
- FLOAT
- STRING

Variables can be single quantities or arrays. **ServoBASIC** ***Plus*** **requires user-defined variables to be declared prior to their use in the program, using the DIM statement. The DIM statement(s) must be the first line(s) of the program (except for comments).**

User-defined variable names must be alphanumeric, less than 40 characters in length, and contain no spaces. They must begin with an alphabetic character. Variables must be declared using the DIM statement prior to their use within a program, otherwise a compilation error will occur.

Note: *Variable names are not case sensitive.*

User-defined variable table

Variable	Declared Type	Representation
INTEGER	INTEGER or LONG	32 bit signed number
FLOAT	SINGLE, DOUBLE, or FLOAT	single precision IEE floating point
STRING	ASCII character strings	

4.1.4 Characters

In addition to the ServoBASIC *Plus* instructions, use alphabetic and numeric characters to create the program, as follows:

Alphabetic Any alphabetic character is legal in ServoBASIC *Plus*. Program instructions are not case sensitive. Alphabetic characters may be typed in either upper or lower case. All text is processed in upper case after compilation. The drive does recognize case when the text is part of a string, that is, text bracketed by quotes for printout or display.

Numeric The digits 0 through 9 are legal for use in ServoBASIC *Plus.*

4.1.5 Operators Used in Programming

The operators used by ServoBASIC *Plus* are *arithmetic*, *relational* and *logical* and are evaluated in that order of precedence. However, operations within parentheses are performed first. Inside the parentheses, the usual order of precedence occurs.

Arithmetic The arithmetic operators are:

Arithmetic Operator	Description of Operation	Example
^	Exponential	2^16
- (one variable)	Negation fo value	- 3
*,/	Multiplication/Division	4.21*3,10.5/2
MOD	Modulus (remainder)	5 MOD 2
+,- (two variables)	Addition/Subtraction	27 + 8, 19 - 2

Note: *When multiple arithmetic operators are used in an expression, they are performed in the order of precedence given in the table; that is, multiplication is performed before addition, and so on. Also, integer division is not supported.*

*For instance, in the expression Value = 2 + 3 * 5, Value is equal to 17. You may use parentheses to set the order of precedence you desire. In the expression Value = (2 + 3) * 5, the parentheses direct 2 + 3 to be solved before the sum is multiplied by 5 to produce a result of 25 for Value.*

Relational

Relational operators are used in `IF-THEN-ELSE`, `WHILE-WEND`, and `FOR-NEXT` statements. The relational operators are:

Note: *Arithmetic operators are performed before relational operators in an executing program line.*

Relational Operator	Description of Operation	Example
=	Equality	IF Value = 9 THEN GOTO LABEL_1
< >	Inequality	IF Value < > 9 THEN GOTO LABEL_1
<	Less than	IF Value < 99 THEN GOTO LABEL_1
>	Greater than	IF Value1 > Value29 THEN GOTO LABEL_1
<=	Less than or equal to	IF Value1 <= Value2 THEN GOTO LABEL_1
>=	Greater than or equal to	IF Value1 >= Value2 THEN GOTO LABEL_1

Logical

Logical operators are used in IF-THEN-ELSE, WHILE-WEND, and FOR-NEXT statements. The logical operators are:

Logical Operator	Description of Operation	Example
AND	Both conditions must be true	IF Value1 > 5 AND Value2 <= 3.00 THEN GOTO LABEL_1
OR	Either or both conditions must be true	IF Value1 = 1 OR Value2 = 0 THEN GOTO LABEL_1
XOR	Either but **not** both conditions must be true	WHILE Value1 > 1 XOR ENCDR.POS = 102400

Note: *Logical operators are performed in the order of precedence given in the table.*

4.1.6 Labels

Labels reference particular lines within a program to permit access by program flow control statements, such as the `GOTO` statement. Labels must be alphanumeric and less than 40 characters in length.

Labels must begin with an alpha character and must be terminated with a colon.

Note: *Labels are NOT case sensitive.*

Example

The following program performs a simple loop, printing the same message over and over and over until the program is terminated.

```
Loop_Again:
  PRINT “All work and no play”
GOTO Loop_Again
```

4.2 Notation Conventions Used in this Manual

The following notation conventions are used in this manual when explaining ServoBASIC *Plus* language use.

Notation	Named	Indicates
[ ]	square brackets	the entry within the brackets is optional
...	3 dots	the entry may be repeated multiple times
`info`	information in computer typeface	computer information displayed on the computer screen
Italics	italicized information	for emphasis or definition

4.3 ServoBASIC *Plus* Instruction Types

Introduction

ServoBASIC *Plus* consists of programming statements or functions, and arithmetic operations permitted in the BASIC programming language. A complete list of these instructions is given in the "SC750 ServoBASIC Plus Reference Manual."

Statements

Statements are of two types, BASIC and PacSci ServoBASIC Plus:

- BASIC statements control the flow of instructions within a program. They direct the execution of functions, for example, comparing function results and going to specific points in the program based on the comparison, prompting for input, printing results of functions, and so on.
- PacSci ServoBASIC Plus statements control the motion of the motor in real time. Motion statements command the motor to move to a specified position or at constant velocity, display parameters and status of the drive, etc.

Functions

BASIC functions *perform a computation and return a value* that can be used in arithmetic expressions. For example, BASIC functions convert decimal numbers to integers, and convert an ASCII code to its equivalent screen display character. PacSci ServoBASIC *Plus* also supports string manipulation functions.

Pre-defined Variables

The SC750 contains a large number of pre-defined variables which are used to extend the capabilities of the standard BASIC language to make ServoBASIC *Plus* capable of motion and I/O control. These variables may be functionally grouped as follows:

- Variables which control functionality of ServoBASIC *Plus* motion and I/O statements. `RUN.SPEED`, `PAUSE.TIME` and `ACCEL.RATE` are examples of this type of predefined variable.
- Variables which control motion and I/O directly. `GEARING`, `ENC.OUT`, and `OUTPUTS` are examples of this type of predefined variable.
- Variables that are maintained by the internal firmware and contain information about the present state of the controller. These variables are typically read-only and include `VELOCITY`, `POSITION`, `MOVING`, and `INPUTS`.

Non-volatile Parameters

There is a relatively small subset of the predefined variables whose values are stored in the non-volatile memory of the SC750 controller. These predefined variables are referred to as non-volatile parameters. These variables, like `KVP` and `KPP` appear at the beginning of a ServoBASIC *Plus* program between PARAMS START and PARAMS END.

Predefined variable types

Variables are *the values acted upon by functions*, or as the result of arithmetic operations. Variables can be further categorized as Read/Write (R/W) or Read Only (R/O). Pre-defined variables are *reserved for use with specific PacSci functions.* These pre-defined variables are either:

- *Floating points* — numbers with values to the right of the decimal place. Used with functions that require decimal numbers, for example the `VELOCITY` variable contains the motor speed in revolutions-per-minute.

or

- *Integers* — integers used with functions that require integers, for example the number of steps to move the motor. Some pre-defined variables are read-only (they cannot be altered from the keyboard or by the program). The `INPUTS` variable, for instance, is dependent solely on the state of the programmable inputs at the connector interface and cannot be altered from the keyboard.

4.4 ServoBASIC *Plus* Functionality

Introduction

This section will describe the details of the ServoBASIC *Plus* programming language. Its organization is broken down into:

- A description of the elements of BASIC used by ServoBASIC *Plus*
- ServoBASIC *Plus* motion control instructions
- Additional ServoBASIC *Plus* language enhancements
- Instructions supporting hardware/software interfaces

4.4.1 Basic Language Instructions

ServoBASIC *Plus* is based upon the BASIC programming language. The key program control statements and functions defined for BASIC are also utilized in ServoBASIC *Plus*. The instructions common to both BASIC and ServoBASIC *Plus* are functionally equivalent. ServoBASIC *Plus* provides enhancements to the BASIC language instruction set which permitting motion control work within the framework of the BASIC language programming structure.

4.4.1.1 BASIC Language Statements

Program flow and decision making control instructions are performed by BASIC language statements. Arithmetic and logical expressions are also valid elements of BASIC language statements. Although a BASIC language programming guide is the best reference for BASIC language programming, the key statements supported by ServoBASIC *Plus* are summarized in the following table:

BASIC language statements

Instruction	Description
BEEP	Transmits a speaker beep command to the serial port.
DIM	Declares a variable type.
CALL SUB	Calls a subroutine, executes it, and returns.
END, END IF, END SUB	Terminates the execution of a program or block structure.
FOR...NEXT	Allows a series of statements in a loop to be executed a specified number of times.
GOSUB...RETURN	Branches to a subroutine, executes it, and returns to instruction following the GOSUB statement.
GOTO	Branches unconditionally to specified label and commences execution.
IT...THEN...ELSE, ...ELSE IF, END IF	Permits conditional execution pending evaluation and outcome of boolean expression.
INPUT	Reads a character string received by the serial communications port.
PRINT	Displays output on the terminal screen while the program is running.
REM or ‘(apostrophe)	Includes comments in the program.
STOP	Stops the executiono f the program.

BASIC language statements

Instruction	Description
SWAP	Exchanges the value of two variables.
WHILE...WEND	Executes a series of statements in a loop as long as the outcome of a boolean expression is true.

4.4.1.2 BASIC Language Functions

Introduction

BASIC functions consist of two fundamental types:

- arithmetic functions
- string functions

Either type performs an operation on a specified argument and returns a result. Arithmetic functions perform a numerical calculation on an argument and return a numerical result. String functions operate on character string arguments. BASIC functions can be incorporated in expressions.

Supported arithmetic functions

Function	Description
ABS( )	Converts the associated value to an absolute value.
ATAN( )	Returns the arctangent of an angle, in radians.
CINT( )	Rounds argument to the next round integer.
COS( )	Returns the cosine of the angle, in radians.
FIX( )	Returns truncated integer part of argument.
INT( )	Converts a variable's value to an integer.
LOG( )	Returns natural logarithem of expression.
LOG10( )	Returns base 10 logarithm of expression.
SGN( )	Returns sign of argument.

Supported arithmetic functions

Function	Description
SIN( )	Returns the sine of an angle, in radians.
SQR( )	Returns square root of an expression.
TAN( )	Returns the tangent of an angle in radians.

Supported string functions

String function	Description
ASC( )	Returns a numeric value that is the ASCII code for the first character of the string.
CHR$( )	Converts an ASCII code to its equivalent character.
HEX$( )	Converts a long integer to a hexadecimal ASCII string.
INKEY$( )	Returns one character read from the serial input port's buffer.
INSTR( )	Provides the location of a substring within a string.
LCASE$( )	Converts a string variable to the lower case value.
LEFT$( )	Returns a string of the leftmost characters of the string.
LEN( )	Returns the number of characters in the string.
LTRIM$( )	Trims leading and trailing spaces.
MID$( )	Returns a substring of a string expression that begins at the specified offset location.
RIGHT$( )	Returns the rightmost characters of the string.
SPACE$( )	Returns a string of spaces.

Supported string functions

String function	Description
STR$( )	Returns a string representation of the value of a numeric expression.
STRING$( )	Returns a string of common charcters.
UCASE$( )	Converts a value to upper case.
VAL( )	Returns the numerical value of the string.

4.4.1.3 BASIC Language Variables

Variables can be single quantities or arrays. ServoBASIC *Plus* requires user-defined variables to be declared prior to their use in the program, using the DIM statement.

User-defined variables must be defined using the **DIM statement.** DIM statements must also be defined on the first line of a program. Furthermore, they must also be less than 40 characters and start with an alphabetic character.

4.4.2 ServoBASIC *Plus* Motion Control

ServoBASIC *Plus* Motion Control consists of the following elements:

- Motion variables
- Motion commands
- Enable and fault reporting
- Encoder input and electronic gearing functions
- Encoder output
- Registration
- Interrupts

4.4.2.1 Motion Variables

Introduction

Before motion control command statements are executed, software variables should be preset to constrain the desired motion profile. These variables specify the commanded motor acceleration and velocity.

Note: *Actual motion dynamics are affected by motor capability, high voltage DC BUS levels, power control current capability, load/motor sizing, drive train dynamics and servo tuning.*

Acceleration/ deceleration

In ServoBASIC *Plus* acceleration is when the magnitude of velocity increases (regardless of motion direction). Deceleration is defined when the magnitude of velocity decreases. Commanded acceleration and deceleration are controlled by `ACCEL.RATE` and `DECEL.RATE`, respectively. Acceleration and deceleration variables are always positive quantities. `ACCEL.RATE` and `DECEL.RATE` are independent.

Scale factor

The scale factor of these variables is 1 Unit = 1 RPM/Second.

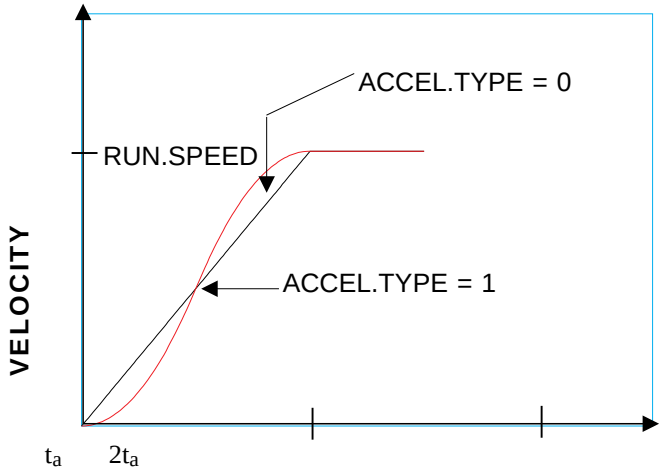
Range

The range of the acceleration type varaibles is 1 to 1,000,000 RPM/Second.

Acceleration type

Two primary types of acceleration are supported by the SC750 servocontroller. These will determine the use of either S-Curve or constant-acceleration velocity ramp profiles indicated on the following graph.

S-Curve profile



$$t_a = \frac{RUN.SPEED}{ACCEL.RATE}$$

S-Curve acceleration

S-Curve acceleration permits a smoother application of motor torque. This may be very desirable for processes moving components that may become uncoupled from the primary power drive train. S-Curve motion commands average acceleration and deceleration rates are defined by `ACCEL.RATE` and `DECEL.RATE`.

Note: *The peak S-Curve acceleration and deceleration rates will be twices the average values.*

When selecting acceleration rates for S-Curve profiles, insure that the motor/controller can attain the peak acceleration rates (twice the averge rates defined by ACCEL.RATE and DECEL.RATE).

Acceleration type is selected with the `ACCEL.TYPE` variable as follows:

- Setting `ACCEL.TYPE = 0` selects constant acceleration/ deceleration ramps.
- Setting `ACCEL.TYPE = 1` selects S-Curve acceleration/ deceleration ramps.

Velocity

Motor velocity is controlled by the RUN.SPEED variable. RUN.SPEED describes the magnitude of motor velocity, hence it is always a positive quantity. Direction is controlled by the DIR variable or by the move distance variable. During velocity commands this constraint sets the peak motor velocity. During positioning motion commands RUN.SPEED sets and limits the maximum commanded motor velocity.

Scale factor

RUN.SPEED is scaled such that 1 unit is 1 revolution-per-minute (RPM) of motor rotation.

Range

The range of RUN.SPEED is 0.0001 to 12,000 RPM.

Velocity mode direction

The DIR variable defines the direction in which the motor will turn when executing a velocity mode command using a GO.VEL statement.

- Setting DIR equal to 0 causes the motor to rotate clockwise, looking at the motor from the shaft end.
- Setting DIR equal to 1 commands the motor to spin counter-clockwise, looking at the motor from the shaft end.

Position

Variables specifying motor position are based on the resolver orientation which is mechanically coupled to the motor shaft. Resolver resolution defaults to 4096 units (or steps-per-revolution).

The distance which the motor will rotate during an incremental index is controlled by the INDEX.DIST variable. The direction of rotation is determined by the sign of INDEX.DIST:

`INDEX.DIST > 0`	Motor rotates clockwise
`INDEX.DIST < 0`	Motor rotates counter-clockwise

TARGET.POS defines the desired position when performing a GO.ABS, an absolute position move command. TARGET.POS can be set to both positive and negative values. The direction of motion performing a GO.ABS command will be determined as:

TARGET.POS > POSITION Motor rotates clockwise

TARGET.POS < POSITION Motor rotates counter-clockwise

Velocity variable

Motor velocity is defined as VELOCITY. This indicates the motor's rotational velocity in RPM, and is read-only. Velocity is a signed float variable containing both magnitude and direction. It is the average over a 128 millisecond interval.

Position variables

All position variables are defined using the same scaling as the position variables, 4096 steps-per-revolution. The range is variable dependent.

RESPOS

The mechanical orientation of the resolver to that of the motor case is indicated by RESPOS. This variable is the basis of motor positioning. Range is 0 to 4096 resolver steps, and this variable is read-only.

POS.COMMAND

The command to the position servo loop is described by POS.COMMAND.

When performing a position move command, this variable is controlled by the desired target position, or index distance, as well as the velocity and acceleration constraints. Position command can be written to, effectively re-establishing the current position. POS.COMMAND can be set to a value, or set equal to zero to establish an electrical home position. Range of this variable is -134,217,728 to +134,217,727 steps (± 32,727 motor revolutions).

POSITION

The feedback signal from the resolver is processed and tracked by the variable POSITION. POSITION defines the actual motor displacement from the electrical home position. The electrical home position is defined as the position where POS.COMMAND equals zero. POSITION will change when POS.COMMAND is changed, but will be directly written to.

POS.ERROR

The difference between commanded position (POS.COMMAND) and actual position (POSITION) is maintained in the variable POS.ERROR. This variable is read-only.

4.4.2.2 Motion Commands

Introduction

All motion commands require the drive to be enabled in order for the motion statement to be executed. If the drive is not enabled when the motion statement is executed, the motion statement will be ignored.

There are three fundamental types of commanded motion. These types and the associated motion command statement for each are described below:

Motion type table

Motion Type	**ServoBASIC *Plus* Statement**
Turn motor shaft at constant velocity, specified by RUN.SPEED and DIR.	GO.VEL
Move motor shaft an incremental index from the current position, specified by INDEX.DIST.	GO.INCR
Command motor to absolute position, specified by TARGET.POS.	GO.ABS
Command the motor to the electrical home position.	GO.HOME
Stop all motion being commanded.	ABORT.MOTION

Motion constraints

Each motion type constrains the commanded motion according to the motion variables `DIR`, `ACCEL.TYPE`, `ACCEL.RATE`, `DECEL.RATE`, and `RUN.SPEED`. The following chart summarizes the variables that must be preset prior to issuing a motion command.

MOTION VARIABLES

MOTION TYPE	SERVOBASIC PLUS STATEMENT	ACCEL.TYPE	ACCEL.RATE	DECEL.RATE	RUN.SPEED	DIR	TARGET.POS	INDEX.DIST	GEARING	ENC.IN	RATIO	PULSES.IN	PULSES.OUT
CONSTANT VELOCITY	GO.VEL	✓	✓	✓	✓	✓							
ABSOLUTE MOVE	GO.ABS	✓	✓	✓	✓		✓						
INCREMENTAL MOVE	GO.INCR	✓	✓	✓	✓			✓					
HOMING MOVE	GO.HOME	✓	✓	✓	✓								
UPDATE PARAMETERS	UPD.MOVE		✓	✓	✓	✓*							
ABORT MOTION	ABORT MOTION												
ELECTRONIC GEARING									✓	✓	✓	✓	✓

* Velocity moves only

Velocity motion commands

During velocity type motion commands the `ACCEL.TYPE`, `ACCEL.RATE`, `DECEL.RATE`, `DIR` and `RUN.SPEED` variables should be set prior to issuing the `GO.VEL` statement. The commanded rate of change velocity is constrained by the acceleration variables. Once the velocity command attains the desired `RUN.SPEED` the servocontroller will maintain this velocity.

Positioning motion commands

During positioning type motion commands, the servocontroller generates a motion command which is constrained by motion variables `ACCEL.TYPE`, `ACCEL.RATE`, `DECEL.RATE`, `RUN.SPEED`, `INDEX.DIST` and `TARGET.POS`. These variables should be set to the desired values prior to issuing the motion command.

Whether an incremental or absolute type positioning command is issued, the servocontroller command is governed as follows:

1. Accelerate to RUN.SPEED according to acceleration variables.
2. Maintain desired RUN.SPEED.
3. Decelerate to zero velocity according to deceleration variables and INDEX.DIST or TARGET.POS.

Note: *Step two, indicated above, will not occur during purely triangular or purely S-shaped velocity profiles. This occurs when RUN.SPEED is higher than that which can be achieved using the specified acceleration constraints.*

Updating motion variables

After issuing, and while currently executing a motion command, the motion variables can be altered but they will have no effect on the currently commanded motion. However, by issuing a UPD.MOVE statement, the ACCEL.RATE, DECEL.RATE, RUN.SPEED and DIR variables can be updated "on-the-fly", that is, without re-issuing a new motion command.

Stopping the motor

There are several ways to stop the motor after a motion statement has been executed.

1. Disable the controller by setting the software enable equal to zero (ENABLE = 0) or by de-activating the hardware enable input (J53-4). This will disable current flow to the motor, thereby generating zero torque. This causes the motor to freewheel or coast.
2. Activation of Directional Inhibit Input signals (J53-1 and J53-2). This will cause the motor to decelerate to zero velocity and disable the generation of torque in the direction for which the input was asserted.
3. Wait for the motion to be completed. (This does not apply to the GO.VEL statement.)

4. Pressing a single Control C while performing program development using Motion Dialogue.
5. Execute an ABORT.MOTION statement. (Program execution continues.)

4.4.2.3 Servocontroller Enable and Fault Reporting

Enabling the power amplifier and the flow of motor current results when three conditions are met. These are:

1) The software variable ENABLE is set equal to 1.
2) The ENABLE input signal (J53-4) is asserted.
3) There are no controller faults present.

Note: *ENABLE defaults to 0 during power up.*

Note: *Disabling the power amplifier does not reduce the high voltage DC bus within the servocontroller chassis.*

Enable variable

Software control of the power amplifier is performed by setting the ENABLE variable. This variable, when set to a logical 1, will enable motor current. Setting ENABLE to a logical zero will disable motor current flow.

Warning
This should not be used as a safety control.

ENABLED variable

ENABLED is a software read-only type variable containing the logical AND of the three power amplifier enable conditions. ENABLED permits user software programs to determine whether the power amplified is activated.

FAULTCODE variable

The controller status display fault code is contained in the program variable FAULTCODE. Controller fault status can be determined within user programs by reading the read-only FAULTCODE variable. The following table indicates the valid FAULTCODE states and the status display on the front panel of the servocontroller.

SC752/SC753

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software Resolver ooverspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	Motor lin-neutral fault
6	6	Control undervoltage
7	7	Bus OV/OC (highest priority)
8	8	No fault, enabled
9	9	Estimated shunt regulator IT fault
b	11	Encoder +5V low
C	12	Teminal +5V low
E	14	Microprocessor fault
F 1	241	Following error overflow
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time rror
F 5	245	PacLAN Error

Status Display	FAULTCODE	Description
F 6	246	Incompatible Motion Dialogue
U C		Unconfigured controller

Note: *Status displays F1, F2, F3, F4, F5, F6 and UC alternately flash between the two values.*

SC754/SC755/ SC756

Status Display	FAULTCODE	Description
0	0	No fault, disabled
1	1	Software Resolver ooverspeed
2	2	Motor overtemperature
3	3	Servocontroller overtemperature
4	4	Servocontroller IT
5	5	Bus overcurrent
6	6	Control undervoltage
7	7	Output overcurrent
8	8	No fault, enabled
9	9	Measured shunt regulator IT fault
A	10	Bus overvoltage or Hot control undervolatge
b	11	Encoder +5V low
C	12	Teminal +5V low
d	13	Power stage control undervoltage
E	14	Microprocessor fault
F 1	241	Following error overflow

Status Display	FAULTCODE	Description
F 2	242	Program memory fault
F 3	243	Parameter memory fault
F 4	244	Run time error
F 5	245	PacLAN error
F 6	246	Incompatible Motion Dialogue
U C		Unconfigured controller

Note: *Status displays F1, F2, F3, F4, F5, F6, and UC alternately flash between the two values.*

4.4.2.4 Encoder Input and Electronic Gearing Functions

Using electronic gearing

Electronic gearing allows you to control the movement of the motor shaft (Slave Axis) with an external encoder (Master Axis) reference. When the shaft of the external encoder turns and electronic gearing has been enabled, then the motor shaft will turn at a precise ratio to the encoder's position.

By processing the external encoder signals, the motor shaft is synchronized with the position orientation of the encoder shaft position.

Motion of the motor (Slave Axis), relative to the reference (Master Axis), is permitted using electronic gearing while executing motion command statements. A potential application for this function would be synchronization of the position of two moving part lines. The slave axis could perform an indexed move to permit the slave axis mechanical phase adjustment.

Connector J52 connections, described in section 2.5.2.1 of the "SC750 Installation and Hardware Reference Manual," indicate proper hookup requirements for external encoder signals.

4.4.2.5 Setting the Electronic Gearing Parameters

ACCEL.GEAR/ DECEL.GEAR

When electronic gearing is turned on, the acceleration and deceleration of the controller will be limited by `ACCEL.GEAR` and `DECEL.GEAR`. This acceleration rate will be used until `GEARLOCK` is achieved. `DECEL.GEAR` should be set before gearing is turned off, as this rate will be used until geared motion has stopped.

ENC.IN

The line count of the external encoder must be set using the `ENC.IN` variable. For instance, a 1000 line encoder requires the user program to execute the following instruction

```
ENC.IN = 1000
```

There are three different variables that you can use to specify the electronic gear ratio. These are `RATIO, PULSES.IN` and `PULSES.OUT`.

RATIO

`RATIO` specifies the electronic gear ratio in terms of motor shaft revolutions to encoder shaft revolutions. The line count of the master encoder must be specified in order to use the `RATIO` variable.

PULSES.IN, PULSES.OUT

`PULSES.OUT` specifies the number of resolver counts the motor will move for a given number of master encoder pulses (`PULSES.IN`) received.

Note: *The last "ratio" variable set determines how the effective ratio is set.*

Example

If `RATIO` is set equal to 3 and then `PULSES.IN` and `PULSES.OUT` are set equal to 2 and 5 respectively, the effective gear ratio will be 2 to 5 because `PULSES.IN` and `PULSES.OUT` were set after `RATIO`.

External encoder input variables

The frequency of the external encoder, in pulses-per-second, is contained in the read-only variable `ENC.FREQ`. To determine the velocity of the external encoder shaft, in revolutions-per-minute, use the following expression:

```
Velocity (RPM) = ENC.FREQ*60/(4*ENC.IN)
```

When the external reference encoder has been connected the encoder position variable (ENCPOS) is updated by the internal software every 1.024 msec. The value of the encoder position is contained in the variable ENCPOS.

This variable continues to be updated even if Electronic Gearing is turned Off. ENC.POS is a read/write variable, hence it can be initialized (set equal to 0) in a program.

Turning electronic gearing On and Off

Electronic gearing is controlled by setting the variable named GEARING. Electronic Gearing can be performed with three different functions. The setting of the GEARING variable is indicated in the following table:

GEARING	Function
0	Gearing off
1	Gearing enabled
2	Follow clockwise master inputs only
3	Follow counter-clockwise master inputs only

Limited Acceleration Gearing

The acceleration and deceleration of the controller, when gearing is turned on or off, will be limited by the software variables ACCEL.GEAR and DECEL.GEAR. When gearing is turned on, the SC750 slave will use the programmed acceleration rates until the slave axis is synchronized with the master axis velocity. This **will** result in the slave axis lagging the master axis position, depending on the master's velocity, specified gear ratio and programmed acceleration rate.

To provide the ability to correct for the position lag when gearing is turned on, there are software variables, GEARERROR and GEARLOCK. GEARERROR contains the amount of accumulated position lag.

When the velocity of the slave is synchronized to the master, the variable GEARLOCK will be set equal to one. An example of the usage of these variables is illustrated in the program below:

```
ACCEL.GEAR = 1200              'Set up Desired Accel.Rate
DECEL.GEAR = 1200              'Set up Desired Decel.Rate
GEARERROR = 0                  'Remove any accumulated error
WHEN INP1=0, CONTINUE          ' Turn on Gearing
GEARING = 1                    ' When INP1 goes low
WHILE GEARLOCK=0 : WEND        ' Wait for Gearlock
INDEX.DIST= GEARERROR          ' Setup for Correction Move
GO.INCR                        ' Perform Phase Correction
```

Note: *The correction move (GO.INCR) will use the acceleration and deceleration variables ACCEL.RATE and DECEL.RATE and the currently specified RUN.SPEED. During the correction move, the maximum motor speed will be:*

$$(\frac{ENC.FREQ \times 15}{ENC.IN} \times RATIO) + RUN.SPEED$$

Note: *Avoid exceeding the base speed of the system.*

When gearing is turned off the slave axis will be decelerated to zero velocity using the variable `DECEL.GEAR`.

4.4.2.6 Encoder Output

The controller's encoder port (connector J52 connections, described in section 2.5.2.1 of the "SC750 Installation and Hardware Reference Manual") may be configured as an encoder output port. This will permit the SC750 series servocontroller to perform as a master in electronic gearing applications. The encoder interface is functionally changed from an input port to an output port under software control using the variable `ENC.OUT`.

If set equal to zero, `ENC.OUT` configures the encoder port as an input, disabling the encoder output transmitter. This is the power up default setting.

To configure `ENC.OUT` as an output port, set it equal to the desired line count of the encoder to be emulated. For example, to emulate a 512 line encoder issue the software instruction:

```
ENC.OUT = 512
```

4.4.2.7 Registration

Registration permits high speed response to discrete input signals. This results in latched resolver and encoder position data. Optionally, an incremental move can be commanded upon receipt of a resolver registration trigger signal.

The SC750 servocontroller features microsecond response to discrete input signals. This allows the controller to support registration functions. The registration trigger signal can be specified from one of three sources.

- Two fast response inputs (Input 7/REG1 and Input8/REG2) provide 5 microsecond response registration. Registration is triggered on the rising edge of the input signals. These signals are input via the J55 connector . For more information, please refer to Section 2.5.2.3 of the "SC750 Installation and Hardware Reference Manual."
- The Z reference channel, or marker pulse, of an external encoder can also be trigger registration. This trigger source provides 1 microsecond response. This input source supports EIA RS-485 differential input signal formats. For more information, please refer to Section 2.5.2.1 of the "SC750 Installation and Hardware Reference Manual."

There are two basic types of registration:

- resolver registration
- encoder registration

Resolver registration

Resolver registration will latch the absolute resolver signal into the variable REG.POS. The input signal that triggers each type of registration is specified by the REG.MODE variable. The resolver position, relative to the motor housing, is latched into the variable REG.RESPOS. Resolver registration can optionally permit an incremental move upon receipt of the trigger input.

Encoder registration

Encoder registration will latch encoder position data into the variable REG.ENCPOS. An incremental move cannot be triggered by encoder registration.

REG.MODE

The variable REG.MODE specifies the registration trigger signal source and the corresponding latched position information. The signal sources for registration are either INPUT/REG1 (J55-7), INPUT8/REG2 (J55-8), or the Z Channel Encoder Reference Differential Input (J52-7, J52-8). The following table indicates how REG.MODE controls the trigger source for latching resolver and encoder position data.

REG.MODE	Resolver Registration Trigger Source	Encoder Registration Trigger Source
	(Latches: REG.POS and REG.RESPOS, Optional Incremental Move)	Latches: REG.ENCPOS
0	Z Channel	Z Channel
1	Input7/Reg1	Z Channel
2	Input7/Reg1	Input7/Reg1
3	Input7/Reg1	Input8/Reg2

REG.FUNC

Registration permits an accurate incremental position to be moved to. This is initiated at the reception of a resolver registration input signal. It occurs if the registration index move is enabled by the control variable `REG.FUNC`.

- If `REG.FUNC` is set to 0, no index move is permitted
- If `REG.FUNC` is set to 1, the motor will displace a distance specified by `REG.DIST` and stop

The currently specified velocity and acceleration variables will be in effect during this command.

Note: *In order to perform an incremental move in response to the resolver registration trigger input, the controller must be enabled, executing a motion command, and in motion. Encoder registration will not trigger an incremental move regardless of the setting of REG.FUNC.*

REG.FLAG

A resettable flag, `REG.FLAG`, permits user programs to determine if a registration input has been activated.

- If the flag is equal to 1, 2 or 3 - a registration input has been received
- If the flag equals 0 - no registration input has been received

Resetting the flag to 0 clears the registration flag and enables the controller to respond to the next registration pulse.

REG.FLAG	Resolver Registration Trigger Occurred	Encoder Registration Occurred
0	No	No
1	No	Yes
2	Yes	No
3	Yes	Yes

Summary of registration usage

1. Specify the registration trigger source using `REG.MODE`.
2. If desired, set the `REG.FUNC` variable to enable an incremental move upon receipt of a resolver registration trigger signal.
3. Reset `REG.FLAG` to enable registration.
4. Receipt of a registration trigger signal can be determined by inspecting `REG.FLAG`.

4.4.2.8 Interrupts

Introduction

Interrupts provide the ability to suspend execution of the main program when a specific interrupt source is being asserted. An interrupt source must be enabled to permit program interruption.

Upon receipt of an (enabled) interrupt, the following events will occur:

- The interrupt source is disabled (received interrupt only)
- The instruction executing when the interrupt occurs is first completed
- The remainder of the interrupt subroutine is executed

  Note: *The interrupt must be re-enabled within the interrupt subroutine, otherwise the interrupt is returned to the main program disabled.*
- Exiting the interrupt subroutine is performed by:
  - Encountering a `RESTART` statement:

    This statement will direct program execution at the start of the main program.
  - Encountering an `END INTERRUPT` clause:

    This designates the (normal) end of the interrupt subroutine. Program execution returns to the instruction following the instruction that was executing when the interrupt was asserted.

Note: *The SC750 servocontroller supports programmable interrupts to permit flexibility in equipment control. However, extreme care must be exercised in implementing interrupts. A thorough understanding of interrupt usage is required.*

Interrupt execution

Execution of an interrupt subroutine requires:

- Definition of the Interrupt Service Routine
- Setting of the Interrupt Enable Flag
- Activation of the Interrupt Signal or Condition

Interrupt sources

Source of interrupts and their associated "Source-Labels" are indicated below:

Interrupt Source	Source Label
Counter-Clockwise Inhibit Active	CCWINH
Counter-Clockwise Over-Travel Liomit	CCWOT
Serial Communications Port	CHAR
Clockwise Inhibit Active	CWINH
Clockwise Over-Travel Limit	CWOT
Disabled	DISABLE
Controller Fault	FAULT
General Purpose Input 1 - High State	I1HI
General Purpose Input 1 - Low State	I1LO
General Purpose Input 2 - High State	I2HI
General Purpose Input 2 - Low State	I2LO
General Purpose Input 3 - High State	I3HI
General Purpose Input 3 - Low State	I3LO

Interrupt Source	Source Label
General Purpose Input 4 - High State	I4HI
General Purpose Input 4 - Low State	I4LO
General Purpose Input 5 - High State	I5HI
General Purpose Input 5 - Low State	I5LO
General Purpose Input 6 - High State	I6HI
General Purpose Input 6 - Low State	I6LO
General Purpose Input 7 - High State	I7HI
General Purpose Input 7 - Low State	I7LO
General Purpose Input 8 - High State	I8HI
General Purpose Input 8 - Low State	I8LO
General Purpose Input 9 - High State	I9HI
General Purpose Input 9 - Low State	I9LO
General Purpose Input 10 - High State	I10HI
General Purpose Input 10 - Low State	I10LO
General Purpose Input 11 - High State	I11HI
General Purpose Input 11 - Low State	I11LO
General Purpose Input 12 - High State	I12HI
General Purpose Input 12 - Low State	I12LO
General Purpose Input 13 - High State	I13HI
General Purpose Input 13 - Low State	I13LO
General Purpose Input 14 - High State	I14HI
General Purpose Input 14 - Low State	I14LO

Interrupt Source	Source Label
General Purpose Input 15 - High State	I15HI
General Purpose Input 15 - Low State	I15LO
General Purpose Input 16 - High State	I16HI
General Purpose Input 16 - Low State	I16LO
PacLAN Error	PacLAN
Position Error	POS.ERROR

Interrupt source descriptions

A brief description of each of the interrupts is provided below.

Interrupt Source	Description	Source Label
Counter-Clockwise Inhibit Active Interrupt	The counter-clockwise inhibit active interrupt will occur when the INH- input is activated. **Note:** *INH- being active may not affect motor motion if the motor is turning clockwise.*	CCWINH
Counter-Clockwise Over-Travel Interrupt	The counter-clockwise over-travel interrupt will occur when the motor POSITION becomes counter-clockwise of the counter-clockwise over-travel limit.	CCWOT
Serial Communications Interrupt	The serial communciations interrupt will occure upon the receipt of a single character via the serial input port. It is recommended that the `INKEY$` string function be used to obtain the input character.	CHAR

Interrupt Source	Description	Source Label
Clockwise Inhibit Active Interrupt	The clockwise inhibit active interrupt will occur when the INH- input is activated. **Note:** *INH- being active may not affect motor motion if the motor is turning counter-clockwise.*	CWINH
Clockwise Over-Travel Interrupt	The clockwise over-travel interrupt will occur when the motor POSITION becomes clockwise of the clockwise over-travel limit.	CWOT
Disabled Interrupt	The disable interrupt occurs when the controller is disabled.	DISABLE
Controller Fault Interrupt	When a controller fault is detected, the FAULT interrupt subroutine may be enabled to perform the desired logic to assist machine control, diagnostics, and fault reporting. (This interrupt source will typically perform an exception to normal motion control conditions. Use fo the RESTART [Label] statement may be the preferred method to exit this interrupt subroutine in oder to configure the main program flow to support this condition.)	FAULT

Interrupt Source	Description	Source Label
General Purpose Input Interrupts (High and Low states)	Interrupt subroutines can be enabled and activated for the General Purrpose Discrete Input 1 through 16. The interrupts can be activated on either the logical high or low signal levels. For example, the source label for an interrupt when Input 3 goes to the logical high signal level is I3HI. The source label for an interrupt when Input 15 goes to the logical low signal level is I15LO.	InHI or InLO
PacLAN Error	A PacLAN interrupt will occur when the drive receives a PacLAN interrupt message over the PacLAN. PacLAN interrupts are generated on other controllers with the LANINTERRUPT[Axis#] statement.	PacLAN
Position Error Interrupt	The position error interrupt will occur when the position error becomes greater thanthe current value of the position error threshold. The current value of the position error threshold is the value of POS.ERROR.MOVING if the motor is moving and POS.ERROR.STOPPED if the motor has stopped. The value of POS.ERROR.MOVING will typically be larger than the value of POS.ERROR.STOPPED.	POS.ERROR

Enabling/ disabling interrupts

Interrupts must be individually enabled (or disabled) to permit (or lock-out) execution of the corresponding interrupt subroutine. This is done by setting the interrupt enable flag corresponding to the interrupt source. The default state for interrupts is disabled. The interrupt enable settings can be polled during program execution to determine which interrupts are enabled.

Disabling interrupts

Disabling (currently enabled) interrupts requires clearing the appropriate interrupt enable flag. The following statement is used to enable or disable interrupts by modifying the interrupt enabled flag for a specific interrupt:

INTR.{Source Label} = x

where:

x = 0	Disable Interrupts
x = 1	Enable Interrupts

Default

x = 0

Note: *When an interrupt is asserted, and the interrupt service routine is executed, the interrupt is disabled. The interrupt service routine must re-issue the interrupt enable instruction to leave the interrupt enabled.*

Examples

The table below contains a few examples using interrupts:

Description of Routine	Instruction
Enable an Interrupt for Input 3 at the logical high state	INTR.I3HI = 1
Enable an Interrupt for an input character via the serial input channel	INTR.CHAR =1
Disable an interrupt when the discrete Input 1 goes to a low logic level	INTR.I1LO = 0

Interrupt subroutines

Interrupt subroutines are written in the same structure as an ordinary subroutine. The interrupt subroutines should be located after the end statement of the main program body. The body of the interrupt subroutine must be structured as follows:

```
INTERRUPT {Source Label}
          .
          .
          .
       User Program Statements
          .
          .
          .
[RESTART ]
END INTERRUPT
```

For example, an Interrupt subroutine that prints a message when a controller fault is detected is programmed as follows:

```
INTERRUPT FAULT
        Print “The controller has faulted”
        Print “The fault code is ”, FAULTCODE
        Print “Correct the fault and press fault reset”
        Input “Press any key to continue”, dummy_variable
RESTART
END INTERRUPT
```

4.4.2.9 Analog Control Blocks (BLKTYPE)

The SC750 can be configured as an analog control block commanded with a signal connected to the analog input channel. The software parameter BLKTYPE specifies the mode of operation as defined by the following table:

BLKTYPE	**Servo Configuration**
0	Torque block - analog command
1	Velocity block - analog command
2	Position loop under software control (default)
3	Position loop - analog command

Note: *The servo control loop block diagrams for the various configurations are included in Appendix A Servo Loop.*

The scale factors associated with the analog input command are defined by the following table:

BLKTYPE	Scale Factor
0	2.00 * CMDGAIN amps/volt*
1	1.00 * CMDGAIN kRPM/V
2	Not applicable
3	0.025 * CMDGAIN revs/volt

* To achieve the indicated scale factor, velocity servo loop gains must be set as follows:

```
KVP = 1.0   KVI = 0
```

The parameter CMDGAIN (default value is defined within the parameter set up program section - typically 1.00) can be modified to customize the scale factor as required.

Note: *Changing BLKTYPE within a program, or from the variables window (when no program is running), will disable the servocontroller. The ENABLE variable must be reset equal to one to permit motor commutation.*

4.4.3 ServoBASIC *Plus* Language Extensions

Introduction

This section describes the following language extensions:

- WHEN statement
- PAUSE.TIME statement
- TIME statement

4.4.3.1 The WHEN Statement

The WHEN statement is used to get extremely fast response to certain input conditions. When the WHEN statement is encountered, the specified condition is evaluated every 0.427 msec and when the condition is satisfied, the specified output action is performed within a millisecond.

If a program encounters a WHEN statement, it will not proceed to the next line of the program until the WHEN condition is satisfied. If the WHEN condition is satisfied and the specified action has been performed, the WHEN statement is complete and program execution continues. In order to repeat evaluation of the WHEN condition, another WHEN statement must be executed.

Satisfaction of the WHEN condition latches the following:

- position command (POS.COMMAND)
- current rotor position (POSITION)
- encoder position (ENCPOS)
- resolver position (RESPOS) into the variables WHEN.PCMD, WHEN.POS, WHEN.ENCPOS and WHEN.RESPOS (within one millisecond).
- RVEL and TIME are latched into WHEN.RVEL and WHEN.TIME.
- WHEN.ANALOG.IN
- WHEN.DACMON
- WHEN.ICMD
- WHEN.IFB
- WHEN.VELCMD

Syntax

The syntax for using the WHEN statement is:

WHEN condition, action

Condition

The condition specifies what condition must be satisfied before the action is performed. The condition may be any one of the following:

- Checking for an Input to be equal to 0 or 1.
- Checking for position command (`POS.COMMAND`) to be greater or less than some value.
- Checking for the motor position (`POSITION`) to be greater than or less than some value.
- Checking for encoder position (`ENCPOS`) to be greater or less than some value.
- Checking the analog input (`ANALOG.IN`) to be greater than or less than some value.

Action

The action specifies what action is to be taken when the condition is satisfied. The action may be any one of the following:

- Setting an Output equal to 0 or 1.
- Setting `RATIO` equal to a new value.
- Performing any one of the following functions:

`GO.ABS`	`GO.HOME`
`GO.INCR`	`GO.VEL`
`PAUSE`	`ABORT.MOTION`
`UPD.MOVE`	

- `CONTINUE` (Allowing program execution to continue with no action performed.)

Example

To rotate the motor at 1000 rpm until Input 3 is pulled low (`INP3 = 0`) and then to decelerate 500 rpm, you can use the following program:

```
RUN.SPEED = 1000

GO.VEL

RUN.SPEED = 500

WHEN INP3 = 0, GO.VEL
```

In this example, Input 3 is checked every millisecond. As soon as Input 3 transitions to a low state (`INP3 = 0`), the program will execute a `GO.VEL` (go at velocity) move.

4.4.3.2 Pausing Program Execution

Encountering a pause instruction suspends program execution until the amount of time specified by the `PAUSE.TIME` variable has elapsed. `PAUSE.TIME` is scaled in seconds. For example, the following instruction sequence will suspend program execution 1.00 second after the `PAUSE.TIME` has been set.

```
PAUSE.TIME = 1.00

PAUSE

.

.

.
```

4.4.3.3 The Time Statement

TIME is a reset table variable containing the current value (in seconds) of a free running timer that is maintained by the internal software. `TIME` has a range of 1 millisecond to 1,833,751 seconds.

The following program will set the TIME variable to zero and print out the value of TIME after 5 seconds.

```
TIME = 0
wait_loop:
IF TIME > 5.0 THEN
     PRINT TIME
ELSE GOTO wait_loop
END IF
```

4.4.4 ServoBASIC *Plus* Interface Instructions

Introduction

This section provides information on the following interface instructions:

- Current limit control
- Discrete I/O ports
- Analog I/O ports
- Serial I/O ports
- Model identification

4.4.4.1 Motor Current Limit Control

Control of peak motor current is facilitated with directional, independent current control parameters. These permit software modification to the available motor torque. Current producing torque in the clockwise direction can be reduced with ILMT.PLUS. Current generating counter-clockwise torque is limited with ILMT.MINUS.

Each parameter permits current adjustability from 0 to 100 percent.

4.4.4.2 Discrete I/O Ports

Inputs

The 16 general purpose input signals are available to be evaluated by programmed logic to perform control functions.

- A low logic level hardware corresponds to a logic 0 input in software.
- A high logic level in hardware results in a logic 1 in software.

The signals can be evaluated individually or through software as composite variables.

The variable `INPUTS` contains the composite decimal representation of all 16 input signals.

- If all 16 input signals are logic 1 (as they will be pulled high if unconnected), `INPUTS` will be 65,535.
- If all the inputs are a logic 0, `INPUTS` will be 0.
- If all the inputs are logic 0, except Input 5, `INPUTS` will be 16.

Individual discrete input channel signal levels can be evaluated by interrogating the INPn variables, INP1 to INP16. They will contain the logic levels of 0 or 1. For instance, if Input 6 is pulled up in hardware to a high logic level, INP6 will be equal to 1.

`INPUTS` and `INPn` variables are read-only.

Event counter

General purpose input 16 serves an alternate function as a hardware event counter input. General Purpose Input 15 functions as the counter's reset signal. The counter can be used to detect an external clock signal. The maximum input frequency is 10 kilohertz. The counter increments once each time the input signal cycles. The current value of the counter is contained in the variable COUNTER. The variable has a range of 0 to 32767. COUNTER can be initialized to a starting value within a program. A low-to-high level signal transition on Input 15 will zero the counter. INP15 can be polled to check this condition.

Outputs

The twelve programmable discrete output signals can be controlled together or individually under software control. Setting an output to a logical level 0 in software, turns the output channel's open collector transistor on, sinking current from the discrete output signal connection to I/O Return.

The variables `OUTn` permit control of individual output signals, `OUT1` to `OUT12`. For example, setting `OUT3` equal to 0 turns on `OUT3`, sinking current to I/O Return. Setting `OUT3` to one turns off the transistor, disabling current flow to I/O return from the `OUT3` signal connection.

The variable `OUTPUTS` permits composite control of all twelve output channels.

- Setting `OUTPUTS` equal to 4095 will turn all the output channels off, simultaneously.
- Setting `OUTPUTS` equal to 0, will turn all the output channels on.
- Setting `OUTPUTS` to 128 will turn Channel 7 on (only).

The discrete output signals are read-write.

Output 12: pulse width modulation (PWM) mode

Programmable Output 12 can be programmed as a PWM type output. The PWM mode will control the duty cycle of an 11.8 kilohertz pulse train. The duty cycle can be varied from 0 percent (Full ON) to 100 percent (Full OFF) by writing to the variable `PWM12`. For instance the statement:

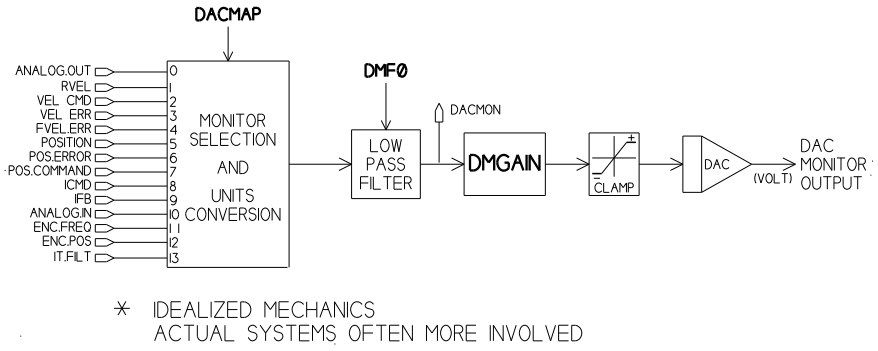
```
PWM12 = 57
```

will output a 57 percent off/43 percent on duty cycle of the 11.8 kilohertz base PWM frequency. The mode of operation, PWM or general purpose output, is dependent on the most recent instruction to program Output12/PWM. Therefore, if the variables `OUTPUTS` or `OUT12` are set, Output 12 will be configured as a general purpose output.

4.4.4.3 Analog I/O Ports

Analog output port

The SC750 provides a user accessible analog output port for servo variable monitoring, or direct user control (ANALOG.OUT). The signal range of the analog output channel is -5.00 to +5.00 volts. Below is a functional diagram of the analog output signal source selection. The control parameter DACMAP selects the desired output signal. The default output test point value for DACMAP is controlled by the servo parameter set up.



The signal selected by DACMAP is passed through a lowpass filter. The filter has a programmable corner frequency, specified by DMF0. The output of the low pass filter, DACMON, is a variable available for monitoring within a program.

DACMAP	Mnemonic	Description	Default Scale Factor
0	ANALOG.OUT	ServoBASIC Variable	1 V
1	VELOCITY	Resolver Velocity	1 V/kRPM
2	VEL.CMD	Net Velocity Command	1 V/kRPM
3	VEL.ERR	Velocity Error	1 V/kRPM
4	FVEL.ERR	Filtered Velocity Error	1 V/kRPM
5	POSITION	Resolver Position	40 V/Rev
6	POS.ERROR	Position Error	40 V/Rev
7	POS.COMMAND	Net Position Command	40 V/Rev
8	ICMD	Torque Current Command	0.5 V/Amp
9	IFB	Measure Torque Current	0.5 V/Amp
10	FAD	A/D After Filter	1 V/V
11	ENC.FREQ	Encoder	0.1 V/kHz
12	ENCPOS	Encoder Position	40 V/4096 Counts
13	IT.FILT	IT Circuit Output	0.5 V/amp

Polarity of the output signal is positive for clockwise torque/velocity/position.

If the selected signal exceeds the $\pm$ 5 Volt range of the analog output channel, the analog output will be clamped approximately at + 5 V or - 5 V.

For position signals, the output signal will “roll over”. For signals of increasing value this means increasing to + 5 Volts then rolling over to -5 Volts and again increasing to +5 Volts. For decreasing position, the output will decrease to -5 Volts, then roll over to +5 Volts and then continue to decrease.

The default scale factors, from engineering units to analog output signal levels are as indicated in the table. Additional modification of the scale factor is permitted using the DMGAIN parameter. DMGAIN functions as a multiplier on the scale factor permitting scale factor increases, decreases, or polarity inversions. Whenever DACMAP is changed DMGAIN is set equal to one. DMGAIN will have no affect on the scaling of the ANALOG.OUT signal (DACMAP = 0).

If DACMAP is set equal to 0, the analog output port can be controlled by writing a decimal value ranging from -5.00 to +5.00 to ANALOG.OUT. This results in an output signal ranging from -5.00 to +5.00 volts. ANALOG.OUT is a read-write variable, and will be updated at the rate in which it is written to.

By configuring DACMAP to output program variables, other than ANALOG.OUT, the analog output channel will be updated every 500 microseconds. The output variables can be additionally scaled by software control using the variable DMGAIN. This permits configuring the output channel to achieve the greatest usable range for the variable of interest.

Analog input port

The analog input signal is sampled every 500 microseconds and the resulting voltage level is stored in the variable ANALOG.IN. The decimal value contained in ANALOG.IN corresponds to the measured dc signal level. The analog input channel has a dynamic range of -12 to +12 volts.

The sampled analog input signal is passed through a lowpass filter. The filter has a programmable corner frequency, specified by ADF0. If the signal source for the analog input is noisy, reduce the corner frequency, as required, to provide additional signal smoothing.

4.4.4.4 Serial I/O Port

The serial communications port is supported by the standard BASIC language input and output instructions. These are summarized in the following table.

Instruction	Description
PRINT	Sends output to the RS-232/RS-485 serial port while the program is running.
INPUT	Obtains information from the RS-232/RS-485 serial port while the program is running.
INKEY$	Returns the character in the serial port input buffer, 0 if none.
CLS	Scrolls data off screen.
BEEP	Transmits a 1/4 second, 800 Hz beep command.

4.4.4.5 Model Identification

The model number of the SC750 is contained in the variable `MODEL`.

4.4.4.6 Firmware Version

The version of the firmware in the controller can be determined by inspecting the variable `FWV` (FirmWare Version).

4.4.4.7 Software Controlled Timers

Six software timers permit activation of a discrete output signal after an discrete input signal has triggered, and a programmable delay has expired. Additionally, the trigger condition can be logically "anded" with a position check output signal. Use of the timers requires specification of the timer variables with a `TMRSET` statement, then enabling the timer.

Timer variable specification syntax is:

TMRSET(timer#n,LEVEL|PULSE,delay_time,OUTn=0|1,INPn=0|1,POUTn=0|1),

where:

timer# — Specifies the timer number. (n is from 1 to 6)

LEVEL|PULSE -Indicates whether the output is to change its output state continuously (LEVEL), or for a fixed duration of time (PULSE).

delay_time — designates the delay time, in second, for LEVEL timers or the pulse duration for the PULSE timer function.

`OUTn`=0|1 — specifies the discrete output to be activated (n is from 1 to 12)

`INPn`=0|1 — denotes the timer trigger input source. (n is from 1 to 16)

`POUTn`=0/1 — determines whether a position check output function is to be logically anded with the timer trigger input source.

Note: *The `POUTn` field must be left blank if not used.*

Disabling or enabling the timers is performed by setting the variable TMENABLEn equal to 0 or 1.

where:

```
TMENABLEn = 0 Disables Timer,
TMENABLEn  = 1 Enables Timer.
```

The state of a timer's output is available in the read-only variable $TMOUT_n$, where n specifies the timer number.

4.5 Servo Tuning and Related Instructions

The SC750 closes velocity and position servo loops within the microcontroller. A block diagram of the servo is presented in Appendix A. Control loop gains and anti-resonant filter adjustments are set with software accessible variables.

Performing the Controller Set Up (part of Options menu) will automatically configure these variables based upon the motor selection and performance requirements.

Note: *Controller set up must be performed prior to running a motor. The controller will not enable the power stage until this is done.*

For most applications, using the "Auto" set up mode will set all servo variables to give adequate performance. If faster response and stiffer positioning is required, select the "Manual" set up mode and use the ↓ key to move to the "System Response" line. Press the space bar until "Stiff/High Bandwidth" is displayed and press ↵. If the Auto and High bandwidth selections produce oscillations, use the space bar to select "Gentle/Low Bandwidth" for the system response. (The low bandwidth option would typically be selected in systems having large inertia mismatches and/or belt coupling between motor and load.) After selecting the appropriate response, follow the prompts to calculate and download the servo variables to the controller. The servo variables can be set individually in manual mode but this is required only in very different situations.

Note: *The servo variables downloaded by the controller set up feature are stored in non-volatile memory and will always be used after power up.*

The variables can be changed by the user program (see description of `ARF0, ARF1, ILC, KVFF, KVI, KVP`, and `KPP` in the "SC750 ServoBASIC *Plus* Reference Manual") but the variables downloaded by the controller set up feature will be restored after power cycling.

5 PacLAN™

Introduction

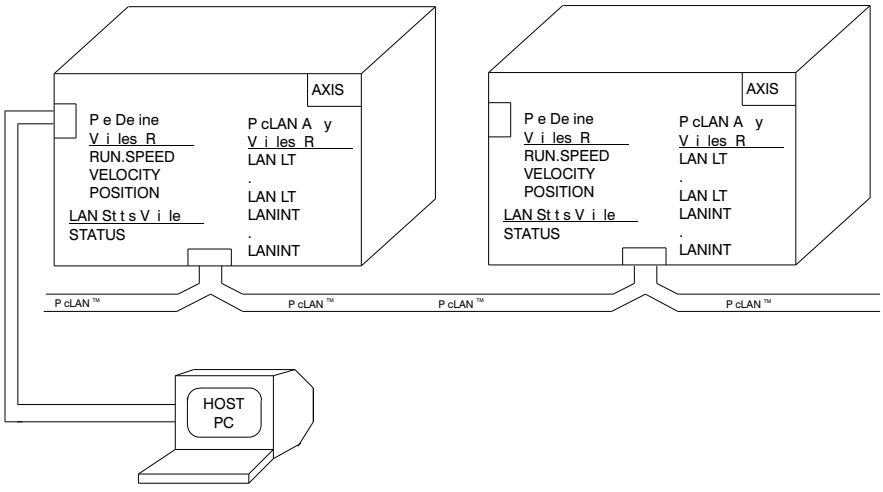
PacLAN is a Local Area Network Port which provides high speed (2.5 MegaBaud) inter-axis serial communication between Pacific Scientific SC750 Servocontrollers. The LAN can provide communications support for as many as 255 SC750 Servocontrollers. Information can be passed between any two axes and is supported by the ServoBASIC *Plus*™ programming language. Program development is supported by the Motion Dialogue Integrated Development Environment.

Implementing PacLAN based inter-axis servocontroller communications requires the following steps:

- Configure each SC750 Servocontroller with a unique PacLAN port address.
- Construct a cable connecting the LAN Ports of the Controllers.
- Develop Programs for each PacLAN controller axis using Motion Dialogue software.
- Implement integrated communications within the servocontroller network.

Note: PacLAN is intended for communications between SC750 Servocontrollers only. Multidrop Serial Communications are not permitted when PacLAN communications are implemented.

PacLAN system diagram



ac

5.1 PC Interfacing

Connections to a PC for program development are implemented by connecting one SC750 of the PacLAN installation to a PC using the RS-232 or the RS-485 serial input/output port of the SC750. The SC750 with the serial port connection functions as the LAN port "Router" to the PC. The Router SC750 controller permits the PC to access the other SC750 controllers via the LAN connection during program development. It is recommended to configure the router axis as address 255, the default RS-232 axis.

Note: The SC750 Router Axis cannot be running a program and communicate with the other SC750s over PacLAN simultaneously.

5.2 PacLAN ServoBASIC *Plus* Language Support

PacLAN provides interaxis communication of ServoBASIC Plus predefined variables and PacLAN array variables. Interaxis variables can be used within a ServoBASIC Plus program in the same manner as local variables. The servocontroller will access variables, over PacLAN , transparently to the user program.

Within a ServoBASIC Plus program, all off-axis variable accesses require the variable name to be appended with the axis address specified in square brackets. Axis designation, with square brackets, is not required for on-axis variable usage. PacLAN provides read-only access to off-axis predefined program variables and a special PacLAN status variable. Each SC750 controller contains two variable arrays which are specifically intended to support inter-axis communication. All PacLAN array variables have read-write capability.

Note: Attempting to read from or write to a controller that is not present on the PacLAN will result in a runtime error. This will cause program execution to terminate and alternating F-4 status display message. Use the PacLAN support variable STATUS[Axis#] to determine if an axis is present on PacLAN without generating a runtime error.

5.2.1 Accessing Predefined Variables PacLAN

PacLAN can access any predefined variable on any other axis. This is performed by appending the axis address (contained in square brackets) to the variable name.

For instance, to set the variable x equal to the motor velocity on axis number three, use the statement:

```
x = VELOCITY[3]
```

To set the target position equal to three times the current target position of axis number 9, use the program statement

```
TARGET.POS = 3 * TARGET.POS[9] .
```

5.2.2 PacLAN Array Variables

Two general purpose read/write variable arrays are available for user defined, inter-axis message passing. One array contains integer variables and the other contains floating point variables.

The integer array variable syntax is designated as:

```
x  = LANINT(n)[Axis#]
```

or

```
LANINT(n)[Axis#] = x,
```

where n is the array subscript, ranging from 1 to 32, and x is an integer quantity. [Axis#] specifies the axis location of the LAN integer variable array.

The floating point variable array syntax is specified as:

```
x = LANFLT(n)[Axis#]
```

or

```
LANFLT(n)[Axis#] = x,
```

where n is the array subscript, ranging from 1 to 32, and x is a floating point value. [Axis#] specifies the axis location of the LAN floating point variable array.

After a LAN array variable write operation is completed, all axes connected to PacLAN will read an updated value for the LAN array variable.

Each array variable is available to every controller interfaced to PacLAN.

5.2.3 PacLAN Status

The read-only software variable, STATUS[Axis#] can be used within a program to determine if an external axis is connected to PacLAN.

If the external axis is present on the network, then STATUS[Axis#] will be set to 1, otherwise it will be set to 0. For example, the status of axis number 3 can be used in a program as follows:

```
IF STATUS[3] = 1 THEN
   PRINT "Axis 3 is on PacLAN"
ELSE
   PRINT "Axis 3 is not connected to
   PacLAN"
END IF
```

Note: Using STATUS[Axis#] to determine if the presence of an axis on PacLAN will not generate a runtime error if it is not connected.

5.2.4 PacLAN Interrupts

Interrupts can be sent from a source to a destination servocontroller axis via PacLAN. To perform this function it is important to insure the destination axis is connected to PacLAN. If this condition is not met, a runtime error will be generated on the axis generating the interrupt. In addition , each controller contains a PacLAN interrupt queue, ten interrupts deep. If the source axis attempts to interrupt a controller with more than ten pending PacLAN interrupts, a run-time error will occur on the interrupt source axis.

An interrupt destination axis must pre-configure a PacLAN interrupt service routine using the same format as previous ServoBASIC Plus interrupt routines. The PacLAN interrupt "{Source Label}" is PACLAN.

For example, the following program statements constitute a valid PacLAN interrupt service routine:

```
INTERRUPT PACLAN
   PRINT "Received LAN Interrupt"
END INTERRUPT
```

The PacLAN interrupt must be enabled using the statement

```
INTR.PACLAN = 1
```

In certain circumstances it may be desirable to include this statement within the interrupt service routine. Note that interrupts are disabled after they are received, until explicitly re-enabled.

The interrupt destination axis can determine the axis address number of the source of the PacLAN interrupt by reading the variable AXIS.INTR.

A PacLAN interrupt is invoked by executing the instruction

```
LANINTERRUPT[Axis#]
```

on the source axis, where Axis# designates the axis address of the controller which is to be interrupted.

5.3 Developing Programs using PacLAN

This section describes special features of the Variables Window when developing programs for PacLAN installations.

Note: The router SC750 controller can not be running a program and perform the "router" function simultaneously.

5.3.1 Running programs out of the variables window

To run a program, that has already been downloaded, to a single PacLAN axis, use the command:

```
RUN[Axis#]
```

To start execution of programs within all PacLAN controllers (all of which have been previously downloaded) issue the command:

```
RUN[0]
```

5.3.2 Stopping program execution from the variables window

To stop a program, executing on an individual PacLAN axis, use the command:

```
STOP[Axis#]
```

To stop execution of all PacLAN connected controller programs, issue the command:

```
STOP[0]
```

5.3.3 Accessing predefined variables from the variables window

By appending the axis address of a PacLAN connected controller, in brackets, as a suffix to a predefined variable name, the predefined variable on the specified axis can be displayed or modified.

6 Servocontroller Initialization

Introduction

The controller set up function of Motion Dialogue provides manual or automatic generation and download of key servo configuration parameters for SC750 Servocontrollers. These parameters specify default controller gains, limits, and constants. The SC750 servocontroller must be initialized prior to operation. Completion of either manual or automatic setup will generate a parameter setup file (DEFAULT.PRM) and a wave shape table for motor commutation (MOTOR#.ITL).

The servocontroller setup parameters are also stored in a PC disk file that can be referenced when creating a program to specify the default configuration for the controller after the program is downloaded. The servo configuration parameters are stored within the controller's nonvolatile memory and once downloaded will become the power-on default values.

Note: *Changing these parameters during program execution is permitted, however, this will not alter the contents of the parameters stored in nonvolatile memory.*

Warning

If the continuous current rating of the drive is greater than the continuous current rating of the motor that it is being used with, then it is possible to cause significant damage to the motor. Pacific Scientific may not honor the warranty of the motor if it is run under these conditions.

6.1 Controller Set Up

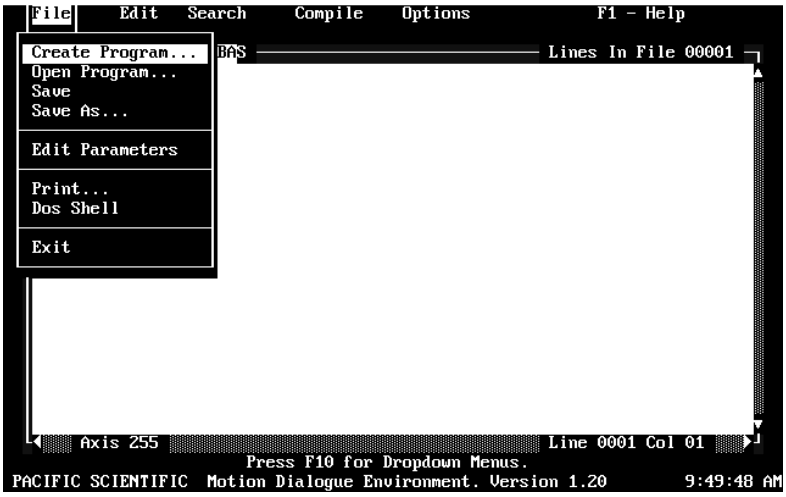
The SC750 controller can be set in two ways:

- Create program
- Controller set up

6.1.1 Create Program

To set up your configuration using the “Create Program” option, you must first enter the file menu.

File screen



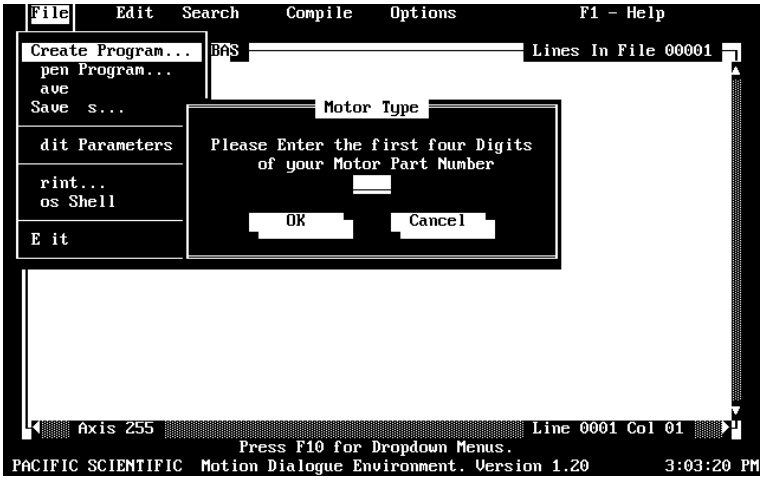
Procedure

Please perform the following procedure to create a program and configure your controller.

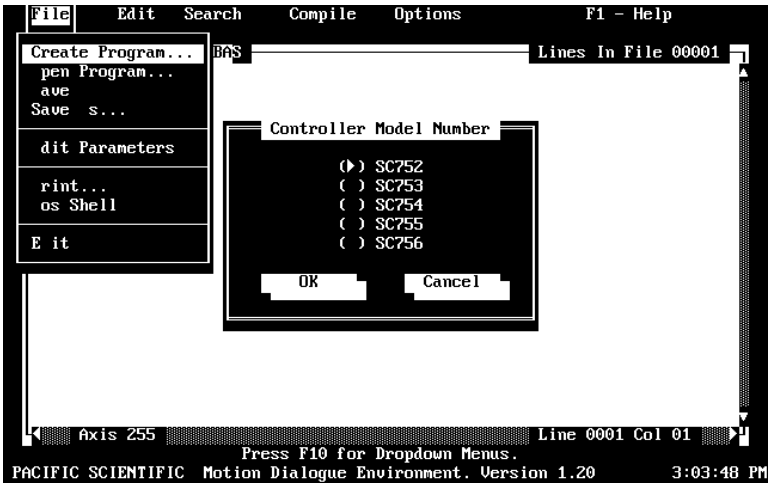
1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (→, ←), move the cursor to highlight “File.”
3. Using the arrow keys (→, ←), move the cursor to highlight the “Create Program” menu selection and press ↵.
4. Motion Dialogue prompts you for the name of the program you are creating. Type in the name, <tab> to OK, and press↵ .

5. You are now prompted for parameter set up information. There are certain parameters needed for any SC750 program. Using the arrow keys (↓,), select “Controller Set Up,” <tab> to OK and press ↵.

6. Motion Dialogue now prompts you for the motor part number. Enter the first four digits of your part number, <tab> to OK and press ↵.



7. You are now prompted for the controller model number. Using the arrow keys (↓,), select the appropriate controller, <tab> to OK and press ↵.



The servocontroller is now accessed via the serial communications link (determined by the model number).

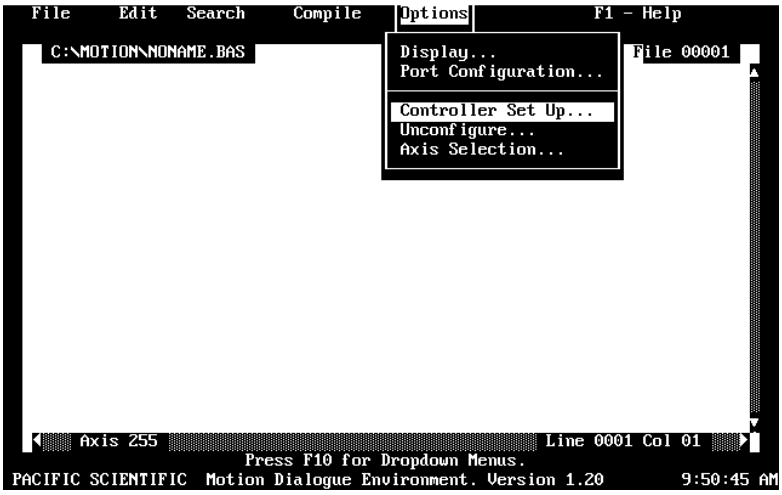
8. The desired set up mode (manual or automatic) can now be selected. (The generation of parameters can be either fully automatic or manual.)

   Note: *Most controller applications can obtain excellent performance using the automatic set up feature.*

6.1.1.1 Automatic Set Up

Fully automatic set up selects servo parameters which should provide stable, medium performance, servo characteristics based on the selected motor and servocontroller combination.

Setup screen



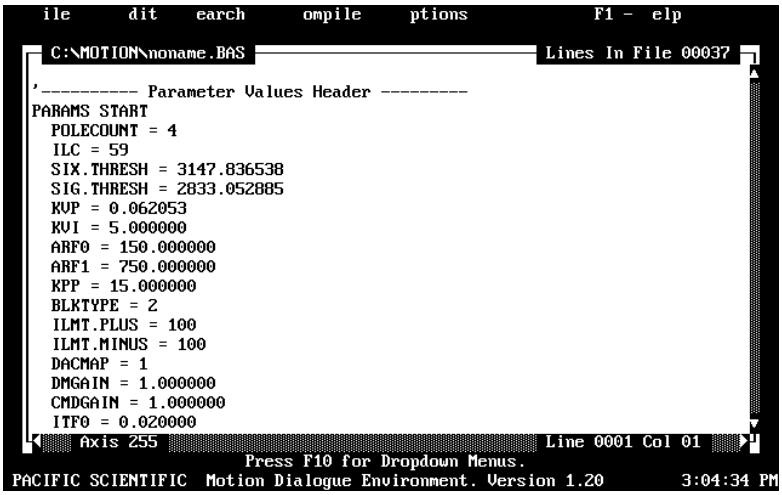
Procedure

To automatically set parameters, perform the following:

1. Using the arrow keys (→, ←) move the cursor to highlight "Auto" and press ↵.

Motion Dialogue will establish values for your servo parameters.

Parameters

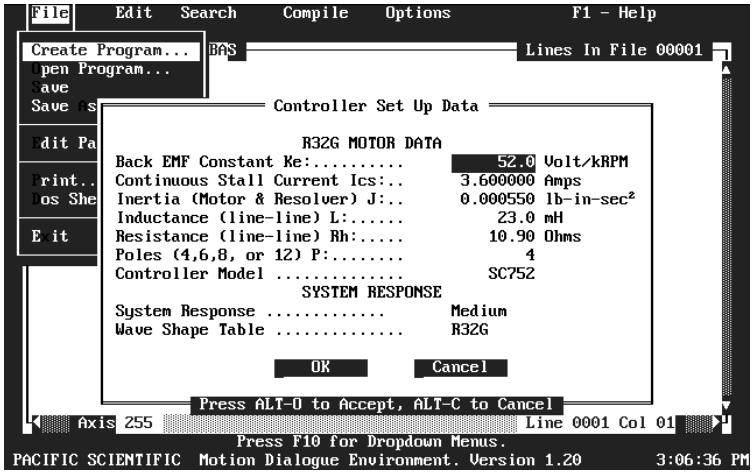


6.1.1.2 Manual Set Up

Manual configuration permits alteration of the motor electrical and mechanical characteristics. The desired servo performance can also be specified using this option.

The default parameters for manual set up are based on the selected motor and servocontroller. The defaults can be modified to customize a specific servocontroller. This is done by editing values within the controller set up data screen.

Manual screen

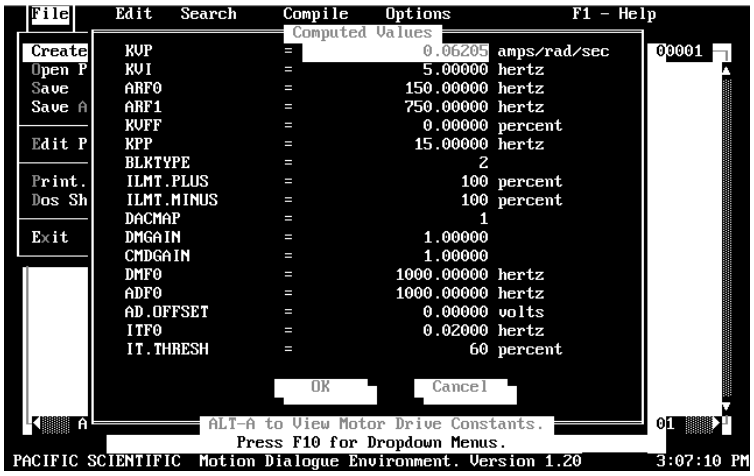


Procedure

To manually set up parameters, perform the following:

1. Using the arrow keys (→, ←) move the cursor to highlight "Manual" and press ↵.
2. The "Controller Set Up Data" screen will be displayed. Using the (↓,) keys highlight the values you would like changed, type in new values and press ↵.
3. After completing changes to the "Controller Set Up Data" screen, <tab> to OK and press ↵.

4. A computed values table will be displayed on the screen.



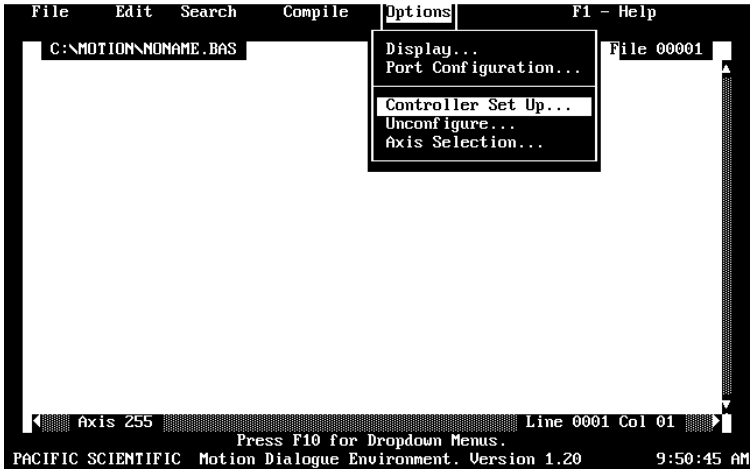
5. Using the (↓,) keys highlight the values you would like changed, type in new values and press ↵.

6. After completing changes, <tab> to OK and press ↵.

6.1.2 Controller Set Up...

The second way in which to configure your controller is through the use of the “Controller Set Up” menu. This is located within the Options menu.

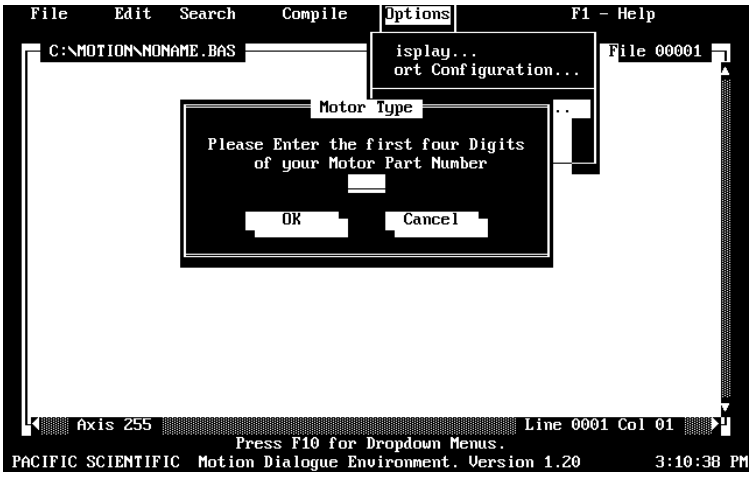
Options screen



Procedure

To set up the controller, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (→, ←), move the cursor to highlight "Options."
3. Using the arrow keys (↓,), move the cursor to highlight "Controller Set Up" and press ↵ (Alternatively, you can type "C" to access "Controller Set Up.")
4. The desired motor part number is prompted for immediately upon entering the Set Up window. Enter the motor part number, <tab> to OK and press ↵.



The servocontroller is now accessed, via the serial communcations link, to determine its model number.

The desired set up mode (manual or automatic can now be selected. (The generation of parameters can be either fully automatic or manual).

Note: *Most controller applications can obtain excellent performance using the automatic set up feature.*

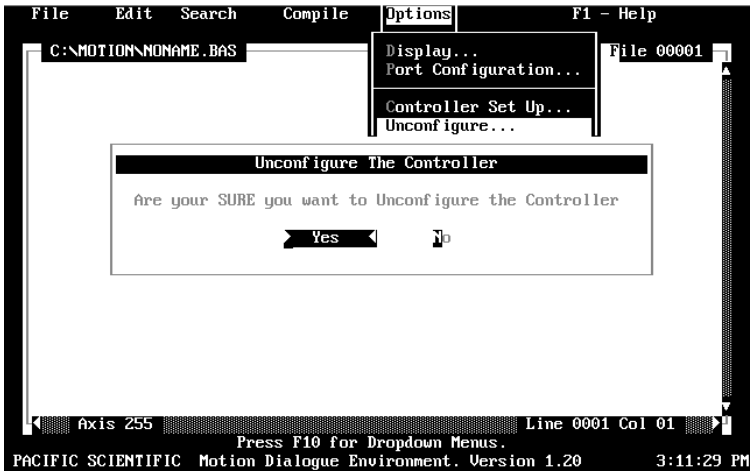
6.2 Unconfigure Controller

Motion Dialogue allows you to remove previously established configuration information.

Procedure

To unconfigure your controller, perform the following:

1. Using the arrow keys (↓,) , move the cursor to highlight "Unconfigure" and press ↵.
2. Motion Dialogue will now prompt with the following, "Are you sure you want to unconfigure your controller?" If yes, <tab> to select OK and press ↵ to continue.
3. To ensure that you do not inadvertently erase the configuration settings, Motion Dialogue will display the screen below. To continue the unconfigure process type "Yes," <tab> to OK, and press ↵.

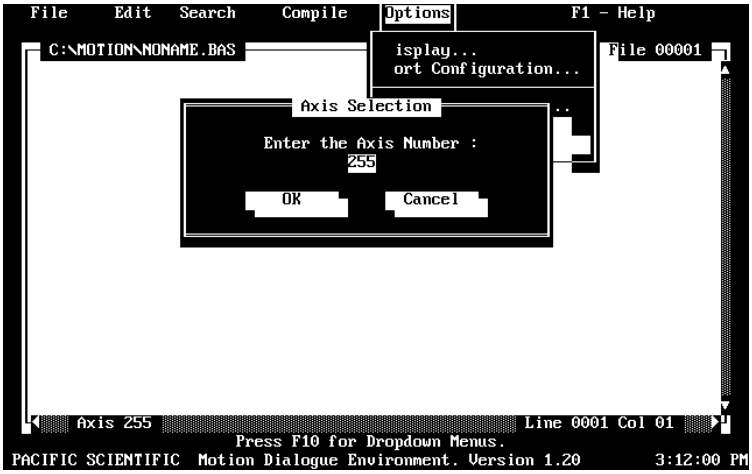


6.3 Axis Selection

The communications address can be established from within Motion Dialogue. The axis address can be set for RS-232 or multidrop communication. For information pertaining to multidrop address values, refer to Section 3.1.1 of the "SC750 Installation and Hardware Reference Manual."

Note: *The default value of the Axis Address is 255. This sets the controller for RS-232 communication.*

Axis selection screen



Procedure

To set the axis address, perform the following:

1. Press F10 to access the pull down menu.
2. Using the arrow keys (→, ←), move the cursor to highlight "Options."
3. Using the arrow keys (↓,), move the cursor to highlight "Axis selection" and press ↵.
4. Enter the desired address value, <tab> to OK and press ↵.

7 Running and Evaluating ServoBASIC *Plus* Programs

In this chapter

The SC750 servocontroller requires a compiled program to be downloaded via serial communications link. Motion Dialogue performs these functions. This chapter describes how to run and evaluate ServoBASIC *Plus* programs. Topics covered include:

- Compile program
- Run program
- Stopping the program
- Breakpoints
- Variable inspection and modification
- Output window
- Real time function calls

7.1 Compile Program

Introduction

The Motion Dialogue complier operates on the ServoBASIC *Plus* program residing in the Edit screen.

Procedure

To compile a program, perform the following:

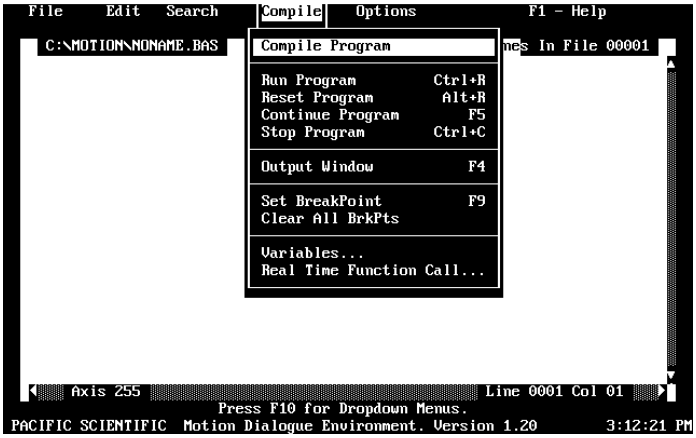
1. Press the **F10 key** to access the pull down menu.
2. Using the arrow keys (→ , ←), move the cursor to highlight "**Compile**."
3. Using the arrow keys (, ↓), move the cursor to highlight "**compile program**" and press ↵ .

Any errors detected during compilation will abort further compilation (after the first error is encountered), display the appropriate error message, then return to the edit screen highlighting the program line containing the error.

Successful compilation will result in the generation of a compiled binary file located on disk, with the base program name of the source file plus a .BIN extension.

Note: *The compiler will compile programs that contain 25,000 bytes of data and 57, 216 bytes of code space. 25,000 bytes total (data **and** code space) can be downloaded to a standard SC750. 25,000 bytes of data space **plus** 57,216 bytes of code space can be downloaded to a "turbo" model SC750.*

Compile menu



Compile menu functions

The table below lists accelerator keys for the Compile functions.

Function	Accelerator Key
Run Program	Ctrl R
Reset Program	Alt R
Continue Program	F5
Stop Program	Ctrl C
Output Window	F4
Set Breakpoint	F9

Note: *Accelerator keys allow you to access a function without using the pull down menu.*

7.2 Run Program

Introduction

Running a program involves downloading a compiled program file to an SC750 servocontroller via the serial communications link. If a program has not yet been compiled, or has been modified since last compilation, the program will be automatically compiled prior to download.

Once the download is complete, Motion Dialogue will automatically enter the output window and monitor serial communications with the servocontroller.

When a program is currently executing within the servocontroller, the decimal point on the status display will be illuminated.

Procedure

To run the currently selected program, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (→ , ←), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Run" and press ↵. (Alternatively, you can type "Ctrl R" to begin running the program.)

Note: *The accelerator key (from within the Edit screen) for the run command is Ctrl R.*

7.3 Stopping the Program

Introduction

Stopping a program is possible by selecting the stop function located within the compile window. This function suspends execution of a program within the servocontroller.

Procedure

To stop program execution, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (→ , ←), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Stop Program" and press ↵ . (Alternatively, you can type "S" to stop the program.)

After a stop execution command has been given, program execution can be continued by selecting "Continue Program" within the compile window, or by pressing the F5 accelerator key.

Note: *Pressing Ctrl C provides the accelerated method of stopping program execution. Suspension of program execution within the servocontroller is indicated by no illumination of the decimal point on the status display.*

7.4 Breakpoints

Introduction

Motion Dialogue can select up to ten breakpoints within a program. Program breakpoints permit evaluation of a program currently downloaded to the controller. Breakpoints should be set prior to running the program. Breakpoints can be set on any executable program line.

Procedure

To run the currently selected program, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Set Breakpoints" and press ↵ . (Alternatively, you can type "S" to set breakpoints.)

Note: *The accelerator key (from within the Edit screen) for the breakpoint command is F9.*

Continuing program execution

When a breakpoint is encountered during program execution the SC750 will suspend program execution and await further instruction from Motion Dialogue.

After a breakpoint has been encountered, program execution can be continued by performing the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Continue Program" and press ↵ . (Alternatively, you can type "n" to continue program execution.)

Note: *The accelerator key (from within the Edit screen) for the continue command is F8.*

Clearing all breakpoints

The compile menu window provides the option of clearing all currently defined breakpoints.

Procedure

To clear defined breakpoints, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Clear All Brkpts" and press ↵ .

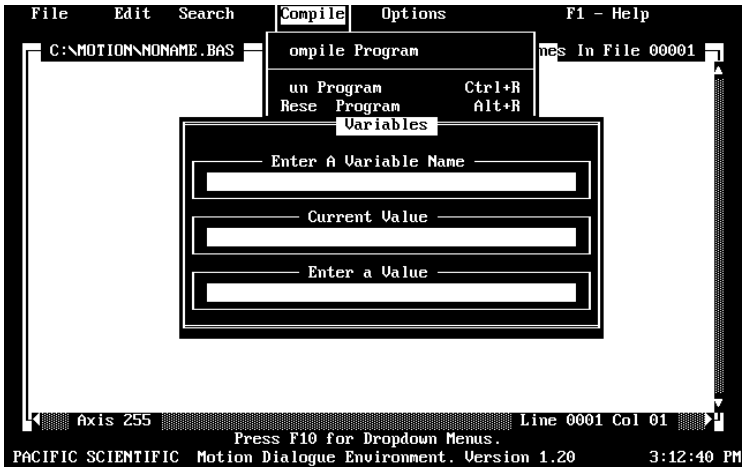
7.5 Variable Inspection and Modification

Introduction

Parameter and variable inspection and modification, is permitted within the Variable window, accessed from within the "Compile Menu."

The variable functions involve uploading and downloading individual variable values from the servocontroller. Variable inspection and modification is permitted only when no program is currently executing in the servocontroller. (i.e. After encountering breakpoints, issuing a stop command, or prior to initial program download.)

Variable window



Inspection procedure

To inspect a variable type, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Variable" and press ↵ . (Alternatively, you can type "V".)
4. Type the variable name in the name entry window and press the ↵ key. (Its value will be shown in the middle display window).
5. To continue updating the variable value, continue to press the ↵ key.

Modification procedure

To modify a variable, perform the following:

1. Press the F10 key to access the pull down menu.
2. Using the arrow keys (← , →), move the cursor to highlight "Compile."
3. Using the arrow keys (, ↓), move the cursor to highlight "Variable" and press ↵ . (Alternatively, you can type "V".)
4. Type the variable name in the name entry window and press the ↵ key. (Its current value will be shown in the middle display window.)
5. Press the down arrow (↓) to move the cursor to the value entry window.
6. Enter the desired value and press ↵ .
7. Press ↵ again, and the updated value will be displayed.

To exit this window press the **ESC** key.

7.6 The Output Window

Introduction

Monitor and control of serial communications during program execution is permitted within the Output window. This window provides communication with the SC750 servocontroller and facilitates checkout of serial communications instructions embedded within the downloaded program. Input and output is performed as though the PC is configured as a terminal emulator.

Note: *Exiting this window during program execution may cause the PC serial port input buffer to overflow.*

To exit this window press the **ESC key** or the **F4 key.**

7.7 Real Time Function Calls

Introduction

Real time control operations can be accessed from within the "Compile Menu." The supported real time functions are:

- GO.ABS
- GO.HOME
- GO.INCR
- GO.VEL
- PAUSE
- ABORT.MOTION
- UPD.MOVE

Real time functions respond the same way as the equivalent ServoBASIC *Plus* instructions. The parameter set up requirements are also the same, and can be performed in the variable window.

Real time functions require transmitting the commands to the servocontroller. Therefore, they are permitted only when no program is currently executing in the servocontroller. (i.e. After encountering breakpoints, issung a stop command, or prior to initial program download.)

To exit this window, press the **ESC** key.

7.8 Power On Program Execution

Once a program has been downloaded to the servocontroller, the automatic execution of the program upon the application of AC control power is specified with the AUTOSTART parameter. This variable can be accessed using the variable inspection and modification window described in section 6.5 of this manual.

AUTOSTART

The parameter AUTOSTART should be set as follows:

- Setting `AUTOSTART = 1` specifies power on program execution.
- Setting `AUTOSTART = 0` requires downloaded programs to be executed using the program run command within Motion Dialogue.

Once a controller has AUTOSTART enabled, and the downloaded program is executed or power is cycled to the controller, a program command must be issued via Motion Dialogue to regain control of the serial interface for further program development and modification.

Note: *If a program is currently executing in the controller, the decimal point in the status display will be lit. To suspend execution issue a Stop Program command, or Ctrl C, from Motion Dialogue.*

8 Program Examples

Introduction This section includes ServoBASIC *Plus* program examples.

8.1 Part Number Program

Background This program provides examples of:

- string manipulation
- array variables
- incremental moves

The program sets up a sequence of moves having parameters determined by a part number entered by the operator using a serial terminal.

The operator is prompted for a part number of form:

PNnn-xx.xx

Parameter n selects the pre-stored run speed, acceleration rate, deceleration rate and dwell time for the part. xx.xx defines the index distance. The part number entered is checked to have exactly the prescribed format.

Data is stored as string variables so that screen displays can be controlled exactly. For example, the length displayed for the part will be exactly as entered and will not have rounding associated with conversions to floating point.

Run speed, acceleration rate, deceleration rate, and dwell time for each part type are stored as arrays which greatly simplifies setting the associated ServoBASIC *Plus* parameters before initiating moves.

The operator is also prompted for the total number of parts required. The parts remaining are displayed while the job is running.

Program example

```
'*************** Program PART_NUM.BAS **************

' ————————Parameter Values Header ————————
PARAMS START
    POLECOUNT = 4
    ILC = 13
    SIX.THRESH = 430.664063
    SIG.THRESH = 387.597656
    KVP = 0.087086
    KVI = 5.0000000
    ARF0 = 150.000000
    ARF1 = 750.000000
    KPP = 15.000000
    BLKTYPE = 2
    ILMT.PLUS = 100
    ILMT.MINUS = 100
    DACMAP = 1
    DMGAIN = 1.000000
    ITF0 = 0.020000
    IT.THRESH = 60
    KVFF = 0.000000
    ADF0 = 1000.000000
    AD.OFFSET = 0.000000
    DMF0 = 1000.000000
    COMMOFF = 0.000000
PARAMS END


'Sets up incremental moves based upon part number entered
'by operator
```

```
'————- Variable Definitions ——————
' for help put cursor on - DIM and hit key
DIM STR1$, STR2$, Type$, Length$, Ans AS STRING
DIM ValidPartNo AS INTEGER
DIM AscCode3, AscCode4, AscCode6 AS INTEGER
DIM AscCode7, AscCode9, AscCode10 as INTEGER
DIM Type AS INTEGER 'value of 3rd & 4th characters of part no

DIM LoopCntr, Selection AS INTEGER

'Machine Parameters:
DIM Pitch AS FLOAT 'Lead screw pitch (in/rev)


'Part Parameters:
DIM Length AS FLOAT
DIM Accel$(5), Decel$(5), Speed$(5), Dwell$(5) AS STRING
DIM Quantity AS INTEGER 'Number of parts in run


'**** Define machine constants and part parameters ****
Pitch = 0.5 'lead screw pitch (inches per rev)
IN.POS.LIMIT = 50 'Position tolerance (rslvr counts)


'Parameters for Part Type 1:
Accel$(1) = "2000" : Decel$(1) = "2000"
Speed$(1) = "500" : Dwell$(1) = "1"


'Parameters for Part Type 2:
Accel$(2) = "10000" : Decel$(2) = "10000"
Speed$(2) = "1000" : Dwell$(2) = ".2"


'Parameters for Part Type 3:
Accel$(3) = "5000" : Decel$(3) = "2000"
Speed$(3) = "800" : Dwell$(3) = ".5"
```

```
‘Parameters for Part Type 4:
Accel$(4) = “20000" : Decel$(4) = ”10000"
Speed$(4) = “1200" : Dwell$(4) = ”.1"


‘Parameters for Part Type 5:
Accel$(5) = “50000" : Decel$(5) = ”50000"
Speed$(5) = “1200" : Dwell$(5) = ”.1"


‘***** Setup prompting for part number ******
GetPartNo:
ValidPartNo = 0
WHILE NOT ValidPartNo
    ValidPartNo = -1
    CLS
    PRINT “ENTER PART NUMBER:”
    PRINT “  PN01-xx.xx”
    PRINT “  PN02-xx.xx”
    PRINT “  PN03-xx.xx”
    PRINT “  PN04-xx.xx”
    PRINT “  PN05-xx.xx”
    PRINT “”
    PRINT “  Where xx.xx is part length in inches”
    PRINT “”
    PRINT “”  INPUT “PART NUMBER”; STR1$

    ‘Check Part Number
    AscCode3 = ASC(MID$(STR1$, 3, 1))
    AscCode4 = ASC(MID$(STR1$, 4, 1))
    AscCode6 = ASC(MID$(STR1$, 6, 1))
    AscCode7 = ASC(MID$(STR1$, 7, 1))
    AscCode9 = ASC(MID$(STR1$, 9, 1))
    AscCode10 = ASC(MID$(STR1$, 10, 1))
    IF LEN(STR1$) < > 10 THEN
        ValidPartNo = 0
    ELSE IF LEFT$(UCASE$(STR1$), 2) < > “PN” THEN
        ValidPartNo = 0
    ELSE IF (AscCode3 < 48) OR (AscCode3 > 57) THEN
        ValidPartNo = 0
```

```
    ELSE IF (AscCode4 < 48) OR (AscCode4 > 57) THEN
        ValidPartNo = 0
    ELSE IF MID$(STR1$, 5, 1) < > "-" THEN
        ValidPartNo = 0
    ELSE IF (AscCode6 < 48) OR (AscCode6 > 57) THEN
        ValidPartNo = 0
    ELSE IF (AscCode7 < 48) OR (AscCode7 > 57) THEN
        ValidPartNo = 0
    ELSE IF MID$(STR1$, 8, 1) < > "." THEN
        ValidPartNo = 0
    ELSE IF (AscCode9 < 48) OR (AscCode9 > 57) THEN
        ValidPartNo = 0
    ELSE IF (AscCode10 < 48) OR (AscCode10 > 57) THEN
        ValidPartNo = 0
    END IF


    Type$ = MID$(STR1$, 3, 2)
    Type = VAL(Type$)  IF Type < 1 or Type > 5 THEN
        ValidPartNo = 0
END IF
    If NOT ValidPartNo THEN
        'If not valid part number prompt operator again
        PAUSE.TIME = 1
        FOR LoopCntr = 1 to 2
            CLS
            PAUSE
            PRINT "INVALID PART NUMBER"
            PAUSE
        NEXT LoopCntr
    ELSE

        'If valid part num, display part parameters to operator    CLS
        Length$ = MID$(STR1$, 6)
        PRINT "Selected Part " + Type$ + " has parameters:"     PRINT "  Acceleration = ";
        Accel$(Type); " RPM/Sec"      PRINT "  Deceleration = "; Decel$(Type); "
        RPM/Sec"      PRINT "  Speed = "; Speed$(Type); " RPM"
        PRINT "  Dwell time = "; Dwell$(Type); " Sec"
        PRINT "  Length = " + Length$ + " In."
```

```
        PRINT ""
        INPUT "Is this OK (Y/N)"; Ans
        IF (Ans < > "Y") AND (Ans < > "y")  THEN
            ValidPartNo = 0
        END IF
    END IF
WEND


'****** Determine number of parts ******
GetQuantity:
CLS
Quantity = 0
WHILE (Quantity < 1) OR (Quantity > 1000)
    INPUT "Number of parts (1 to 1000)"; Quantity
WEND
'** Enable controller.  Signal operator if not enabled **
ENABLE = 1
PAUSE.TIME = .2
WHILE ENABLED = 0
    OUT1 = 0
    PAUSE
    OUT1 = 1
    PAUSE
 WEND


'******** Setup Move Parameters *******
ACCEL.RATE = VAL(Accel$(Type))
DECEL.RATE = VAL(Decel$(Type))
RUN.SPEED = VAL(Speed$(Type))
PAUSE.TIME = VAL(Dwell$(Type))
INDEX.DIST = 4096 * (VAL(Length$) / Pitch)


'***** Run cycles ******
CLS
OUT2 = 0 'Indicate "Machine in operation"
FOR LoopCntr = 1 to Quantity
    GO.INCR
    WHILE IN.POSITION = 0
```

```
    WEND
    'Display moves remaining
    PRINT (Quantity - LoopCntr)
    PAUSE
NEXT LoopCntr
OUT2 = 1


'***** Signal operator we're done *****
PAUSE.TIME = 2
STR2$ = ""
WHILE LEN(STR2$) = 0
    CLS
    PRINT "RUN COMPLETE!"
    PRINT "HIT ANY KEY TO CONTINUE"
    PAUSE
    STR2$ = INKEY$
WEND


'**** Prompt operator for next operation ****
CLS
PRINT "Run more of same part? (1)"
PRINT "Run new part? (2)"
PRINT "Quit? (3)" Selection = 0
WHILE (Selection < 1) OR (Selection > 3)
    INPUT "Selection (1, 2 or 3)"; Selection
WEND
IF Selection = 1 THEN
    GOTO GetQuantity
ELSE IF Selection = 2 THEN
    GOTO GetPartNo
END IF
END 'Main Program
'————————— Subroutines ———————————
'————————— Interrupt Routines ————————-'————————— End of Program Listing
```

8.2 Seek Home Program

Background

This program illustrates the use of a subroutine, SEEKHOME, that emulates the functionality of the SEEK.HOME command of Servo BASIC. This command is not supported in ServoBASIC *Plus* because it is easily implemented using other instructions as shown here. Implementing the home search as a subroutine allows user customization based upon machine requirements.

The demonstration program first enables the controller, then calls the subroutine SEEKHOME, and (if homing is successful) moves to an absolute target position of 2 * 4096. SEEKHOME searches for the home switch, connected to INP1, at a speed and direction specified in the subroutine. The maximum search distance from the starting position is also specified (in this example is is 200 turns).

The polarity of the home switch can be high (1) or low (0) when at the home switch. Once the switch is found, a crawl in the opposite direction is initiated and mechanical home is defined as the position where INP1 transitions. An offset (HomePosOffset) can be defined between the mechanical home position and the "electrical home position" where POS.COMMAND is zero.

The subroutine shown here has more generality than the original SEEK.HOME and can easily be expanded upon by the user.

Program example

```
'*************** Program SEEKHOME.BAS ************'

'* Illustrate subroutine used to find home switch * '

' ————————Parameter Values Header ————————
PARAMS START
    POLECOUNT = 4
    ILC = 13
    SIX.THRESH = 430.664063
    SIG.THRESH = 387.597656
    KVP = 0.087086
    KVI = 5.0000000
    ARF0 = 150.000000
    ARF1 = 750.000000
    KPP = 15.000000
    BLKTYPE = 2
    ILMT.PLUS = 100
    ILMT.MINUS = 100
    DACMAP = 1
    DMGAIN = 1.000000
    ITF0 = 0.020000
    IT.THRESH = 60
    KVFF = 0.000000
    ADF0 = 1000.000000
    AD.OFFSET = 0.000000
    DMF0 = 1000.000000
    COMMOFF = 0.000000
PARAMS END
```

```
'————————— Variable Definitions ——————
'for help put cursor on - DIM and hit  key
DIM HomeSwSearchDir, MaxHomeSwSearch, HomeSwActive as INTEGER   DIM
HomePosOffset, FoundHomeSw, Homed AS INTEGER
DIM HomeSwSearchSpeed,  HomeSwCrawlSpeed, HomeSpeed as FLOAT



PAUSE.TIME = .1
ENABLE = 1
OUT1 = 1
WHILE ENABLED = 0
    OUT1 = 0
    PAUSE
    OUT1 = 1
    PAUSE
WEND
CALL SEEKHOME
IF NOT HOMED THEN
    PRINT "HOMING ROUTINE NOT SUCCESSFUL"
ELSE
    PRINT "HOMING COMPLETE"
    IN.POS.LIMIT = 10
    ACCEL.RATE = 10000
    DECEL.RATE = 10000
    RUN.SPEED = 500
    TARGET.POS = 2 * 4096
    GO.ABS
    WHILE IN.POSITION = 0
    WEND
    PRINT "I'm now at absolute position "; POSITION
END IF
END 'Main Program
```

```
'————————-Subroutines ———————————

'******** Subroutine to search for home switch *********

SUB SEEKHOME


'**** Parameters for homing.  Customize as required. ****
ACCEL.RATE = 10000
DECEL.RATE = 10000
HomeSwSearchDir = 1 'CounterClockwise search
HomeSwSearchSpeed = 200 'Speed when looking for home switch  HomeSwActive = 0
'Home switch pulls INP1 to 0 when active
HomeSwCrawlSpeed = 10
MaxHomeSwSearch = 200 * 4096 'Max distance will move      'during search for switch
HomePosOffset = 0 'Offset between home and home switch
HomeSpeed = 1000 'Speed for move from switch to home


'* Tell world we don't know where home switch is yet**
Homed = 0
FoundHomeSw = 0


'**** Check that system is enabled ****

IF ENABLED = 0 THEN
    GOTO DONE
END IF
```

```
'**** Setup search for home switch ****
POS.COMMAND = 0
RUN.SPEED = HomeSwSearchSpeed
If HomeSwSearchDir = 1 THEN
    TARGET.POS = -MaxHomeSwSearch
ELSE
    TARGET.POS = MaxHomeSwSearch
END IF


'**** Do search ****
GO.ABS
RUN.SPEED = 0
IF HomeSwActive = 0 THEN
    WHILE MOVING
        IF INP1 = 0 THEN
            GO.VEL
            FoundHomeSw = -1
        END IF
    WEND
ELSE
    WHILE MOVING
        IF INP1 = 1 THEN
            GO.VEL
            FoundHomeSw = -1
        END IF
    WEND
END IF
```

```
  '** Crawl in opposite direction for switch transition **
IF FoundHomeSw THEN
    RUN.SPEED = HomeSwCrawlSpeed
    TARGET.POS = 0
    IF HomeSwActive = 0 THEN
        GO.ABS
        RUN.SPEED = 0
        WHEN INP1 = 1, GO.VEL
    ELSE
        GO.ABS
        RUN.SPEED = 0
        WHEN INP1 = 0, GO.VEL
    END IF
    WHILE MOVING
    WEND
    POS.COMMAND = POS.COMMAND - WHEN.PCMD -HomePosOffset
    TARGET.POS = 0
    RUN.SPEED = HomeSpeed
    GO.ABS
    WHILE MOVING
    WEND
    HOMED = -1
END IF


DONE:


END SUB
'————————— Interrupt Routines ——————————
'————————— End of Program Listing —————————
```

8.3 Simplified Seek Home Routines

Background

This program demonstrates two simplified homing routines that can be called from within a main program to establish the electrical home. It is assumed that there is no motion when the homing routines are called. The electrical home is established using input INP1. Position is commanded to the home position before subroutines exit. The main program uses input INP2 to illustrate the homing subroutines.

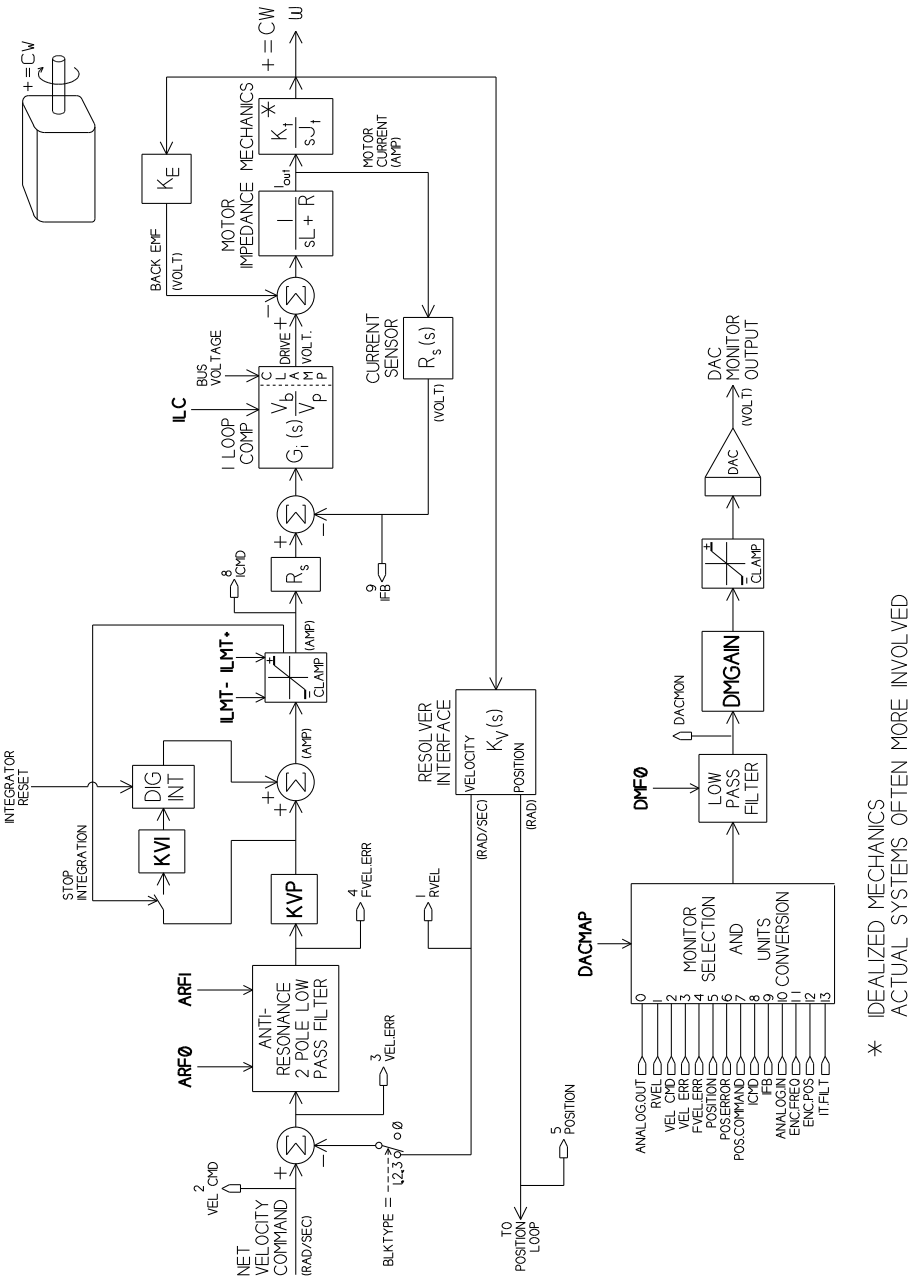
Program example

```
'————————— Parameter Values Header —————————
PARAMS START
    POLECOUNT = 4
    ILC = 13
    SIX.THRESH = 430.664063
    SIG.THRESH = 387.597656
    KVP = 0.087086
    KVI = 5.0000000
    ARF0 = 150.000000
    ARF1 = 750.000000
    KPP = 15.000000
    BLKTYPE = 2
    ILMT.PLUS = 100
    ILMT.MINUS = 100
    DACMAP = 1
    DMGAIN = 1.000000
    ITF0 = 0.020000
    IT.THRESH = 60
    KVFF = 0.000000
    ADF0 = 1000.000000
    AD.OFFSET = 0.000000
    DMF0 = 1000.000000
    COMMOFF = 0.000000
PARAMS END
```

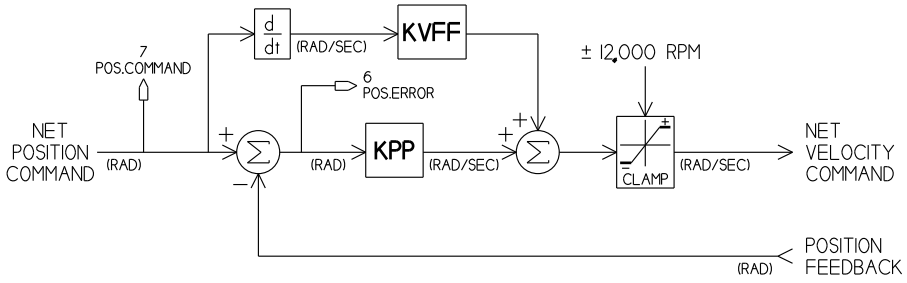
```
‘————— Variable Definitions —————
‘for help put cursor on - DIM and hit  key

‘————— Main Program ———————————
REM The Home switch (INP1) is active on falling edge
‘(Closes).  REM The homing subroutines command motion
‘till switch (INP1)
REM signal goes low, then reverse direction and
‘determine the
REM location of the low-to-high switch signal
‘(Opening Switch) and
REM established electrical home (POS.COMMAND = 0).

‘This main program calls a cw or ccw homing
‘search depending on
‘the state of switch INP2

        if inp2 = 0 then
            call seek_home_cw
            print “clockwise homing complete”
        else
            call seek_home_ccw
            print “counterclockwise homing complete”
        end if
END ‘Main Program

‘————— Subroutines ——————————
REM this routine seeks home in the clockwise direction
sub seek_home_cw
run.speed = 100                     ‘set search speed
dir = 0                             ‘set cw search
go.vel                              ‘command search
run.speed = 0                       ‘preset stop speed
when inp1 = 0, go.vel               ‘wait for falling edge
                                    ‘of switch, then stop
while moving = 1 : wend             ‘complete commanded stop
dir = 1                             ‘set direction toggle
run.speed = 5                       ‘low speed to  backup
go.vel                              ‘issue command to backup
when inp1 = 1, continue             ‘search for switch edge
```

```
pos.command = pos.command - when.pcmd          ' 0 electrical home
go.home                                        'Servo to home position
while moving = 1 : wend                        'complete commanded stop

end sub                                        'back to calling routine

REM this routine seeks home in the clockwise direction
sub seek_home_ccw
run.speed = 100                                'set search speed
dir = 1                                        'set ccw search
go.vel                                         'command search
run.speed = 0                                  'preset stop speed
when inp1 = 0, go.vel                          'wait for falling edge
                                               'of switch, then stop
while moving = 1 : wend                        'complete commanded stop
dir = 0                                        'set direction toggle
run.speed = 5                                  'low speed to  backup
go.vel                                         'issue command to backup
when inp1 = 1, continue                        'search for switch edge
pos.command = pos.command - when.pcmd          ' 0 electrical home
go.home                                        'Servo to home position
while moving = 1 : wend                        'complete commanded stop

end sub                                        'back to calling routine


'————————— Interrupt Routines ———————————
'————————— End of Program Listing ————————
```

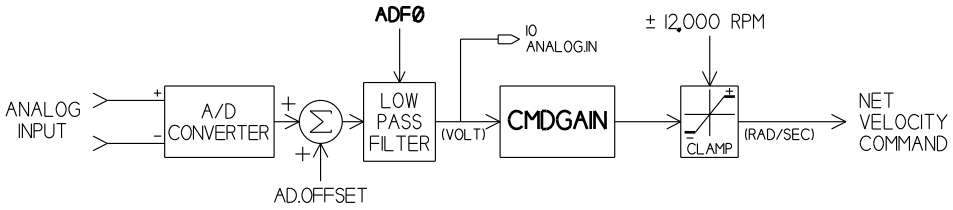
Appendix A Servo Loop



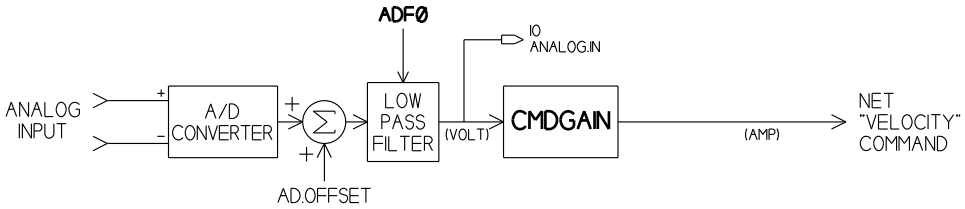
BLOCKTYPE 2: SERVO BASIC POSITION



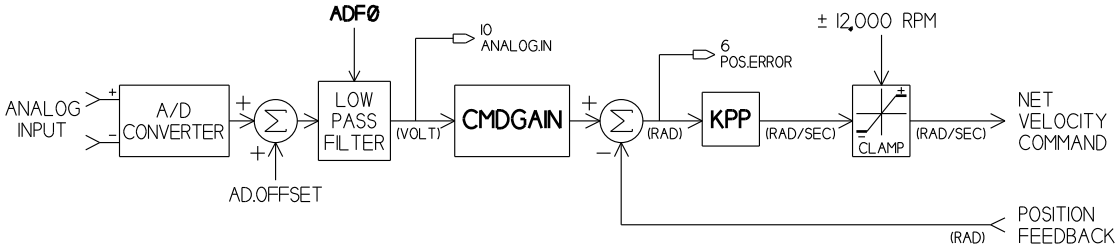
BLOCKTYPE 1: ANALOG VELOCITY



BLOCKTYPE 0: ANALOG CURRENT



BLOCKTYPE 3: ANALOG POSITION



Appendix B ASCII Codes

ASCII Code Result			ASCII Code Result		ASCII Code Result		ASCII Code Result	
0	^@	NUL	32		64	@	96	‘
1	^A	SOH	33	!	65	A	97	a
2	^B	STX	34	\	66	B	98	b
3	^C	ETX	35	#	67	C	99	c
4	^D	EOT	36	$	68	D	100	d
5	^E	ENQ	37	%	69	E	101	e
6	^F	ACK	38	&	70	F	102	f
7	^G	BEL	39	‘	71	G	103	g
8	^H	BS	40	(	72	H	104	h
9	^I	HT	41	)	73	I	105	i
10	^J	LF	42	*	74	J	106	j
11	^K	VT	43	+	75	K	107	k
12	^L	FF	44	,	76	L	108	l
13	^M	CR	45	-	77	M	109	m
14	^N	SO	46	.	78	N	110	n
15	^O	SI	47	/	79	O	111	o
16	^P	DLE	48	0	80	P	112	p
17	^Q	DC1	49	1	81	Q	113	q
18	^R	DC2	50	2	82	R	114	r
19	^S	DC3	51	3	83	S	115	s
20	^T	DC4	52	4	84	T	116	t
21	^U	NAK	53	5	85	U	117	u
22	^V	SYN	54	6	86	V	118	v
23	^W	ETB	55	7	87	W	119	w
24	^X	CAN	56	8	88	X	120	x
25	^Y	EM	57	9	89	Y	121	y
26	^Z	SUB	58	:	90	Z	122	z
27	^[	ESC	59	;	91	[	123	{
28	^\	FS	60	<	92	\	124	|
29	^]	GS	61	=	93	]	125	}
30	^^	RS	62	>	94	^	126	~
31	^_	US	63	?	95	_	127	

Appendix C Multidrop Serial Communications

Note: *The default serial communications format for the SC750 is RS-232. The address value is AXIS 255 (all S1 switches in the UP or OFF position). Multidrop protocol does not apply to RS-232 communications.*

SC750 Multidrop protocol

A multidrop system consists of a multidrop master and 1 to 32 multidrop subsystems. Each subsystem has a unique address ranging from 0 to 254. The address of a subsystem is configured using the dip switch S1, as described in Section 3.1.1 of the "SC750 Installation and Hardware Reference Manual." The subsystem address is indicated in the software variable AXIS.ADDR.

Note: *Only one multidrop subsystem can transmit data back to the multidrop master at any given time.*

Configuring a multidrop subsystem to transmit data to the multidrop master requires the multidrop master to transmit a message which selects the multidrop subsystem as the unique logged on system. A variable indicating the logon status (LOGGEDON) will be set appropriately in all the multidrop subsystems connected on the multidrop interface.

A logged on multidrop subsystem can input received data during program execution using the INKEY$ function or INPUT statement. If subsystems are not logged on, they cannot access data transmitted by the multidrop master. Also, an INPUT statement program execution will effectively be suspended until the subsystem has been issued a logon message and receives valid input data terminated with the carriage return character. The INKEY$ statement will return a null string, with a value of 0, if a subsystem is not logged on.

A logged on subsystem can transmit data to the multidrop master using the PRINT statement while the program is executing. If a subsystem is not currently logged on, its multidrop transmitter is disabled and program execution will not halt at the PRINT statement.

Subsystem selection

A subsystem's address is configured using the S1 dip switch located underneath the small panel on top of the controller. Setting up this switch for a particular address is described in Section 3.1.1 of the "SC750 Installation and Hardware Reference Manual." The subsystem address is indicated in the software variable AXIS.ADDR.

Note: *The subsystem address S1 switch setting is polled only when power is applied to the controller. If the switch setting is modified, then power must be cycled to the controller for the new address to take effect.*

When a multidrop subsystem is logged on, its' multidrop transmitter can be enabled whenever data is to be transmitted. If a unit has not been logged on (since AC power was applied) or if a valid logon command has been issued to another SC750 multidrop subsystem, the subsystem's transmitter is disabled.

The multidrop master must transmit a multidrop subsystem selection, or logon command, message using the format:

/nnn,

This permits the multidrop subsystem's transmitter to be enabled. "nnn" is a valid SC750 subsystem address, ranging from 0 (all switches ON) to 254.

Note: *Address 255, all switches OFF, indicates the unit is configured for RS-232 serial communications.*

Once the "/" (slash) has been transmitted, the subsystem address is defined by the three digit numeric code. If there are less than three numeric digits, the address is terminated by the first non-numeric character.

When a multidrop subsystem that is currently logged on recognizes selection of another subsystem, it will disable its multidrop transmitter.

A logon variable (LOGGEDON) indicates that a multidrop subsystem has been selected to transmit to the multidrop master. This variable will be updated in all multidrop subsystems after the multidrop master has issued the logon message. This variable can be used to determine the status of a multidrop subsystem.

Change of subsystem address

If a multidrop master changes the subsystem address by issuing a new logon command, then a subsystem that was previously logged on will suspend its multidrop transmit and receive functions as follows:

The INKEY$ function will not indicate data characters received by the multidrop interface and will always return a null string (value of zero).

If an INPUT statement is encountered after the address is changed, the subsystem will effectively suspend program execution until it has been re-selected as the multidrop subsystem and the INPUT statement receives valid data terminated with a carriage return. If a logon command is received while an INPUT data message is being received (embedded within the data), the INPUT statement will ignore the data transmitted prior to the logon command, await its subsystem to be addressed with a new logon command, re-prompt with a “Redo from Start” message, and wait for valid data to be input terminated with a carriage return.

The PRINT statement will complete transmission of the most recent (single) character being output to the multidrop transmitter, prior to issuing the change of the subsystem address. The subsystem’s multidrop transmitter will be disabled when subsequent characters are output, however the PRINT statement(s) will continue execution regardless of the logon state.

Due to the potential suspension, or hanging, of program execution if the selected subsystem address is changed while a subsystem is receiving input data, proper synchronization of the master and multidrop subsystems should be carefully developed. You may want to perform software handshaking to support the communications between the multidrop master and subsystems.

RS-232 notes

Serial data from the RS-232 RXD input is wire “or”-ed with the multidrop (RS-485) RXD input channel. Multidrop input communications are not intended to be used simultaneously with RS-232 input sources.

Executing a PRINT statement when a multidrop subsystem is logged off will result in serial data being transmitted on the RS-232 channel only. When the subsystem is enabled, data will be transmitted on both the RS-232 and multidrop (RS-485) output channels.

Index

D

E

F

H

I

L

M

N

O

P

R

S

T

U

V

W