NMC Simple Sequencer
Stand-alone Controller

1.0 Overview

The **Simple Sequencer** is a programmable, stand-alone controller for **NMC** control modules including the **PIC-SERVO**, **PIC-STEP** and the **PIC-I/O**. Sequences of commands are programmed into non-volatile memory using an intuitive Windows interface. No programming experience is required for creating command sequences. Command sequences can be created, edited, downloaded and even monitored from the same simple Windows application. When completed, your program will run automatically on power-up.

Simple Sequencer programs are simply lists of commands to be executed by **NMC** control modules or by the **Simple Sequencer** itself. Program flow can be controlled by branching on switch inputs, motor operating conditions, or on input from a terminal or another computer.

The **Simple Sequencer** programming application runs under Windows 95/98/NT. One display window gives you complete control over programming, editing and interactive running of your programs. Programming consists the following steps:
1. Select an **NMC** module from the module list
2. Select a command from the command list
3. Enter the command parameters as prompted
4. Add the command to the program listing

Once your program is complete, click on the "Load" button, and you are ready to run your program, either interactively or stand-alone.

The **Simple Sequencer** has the following features:
- Plugs into the existing socket on your **Z232-485** converter board
- Programs execute automatically on power-up
- RS485 communication to **NMC** control modules
- RS232 communications with external computers or terminals
- Programs may include branching, subroutines, and automatic monitoring of error conditions
- Programs of up to 2000 commands can be stored

The **Simple Sequencer** is designed to be an easy-to-use, programmable controller for **NMC** control modules. However, it does have some limitations on the complexity of programs which can be written. For applications requiring more complex control, including symbolic variables, math functions, or precise coordination of several modules, you should explore the use of our DOS or Windows programming tools for PC based control, or the use of the C programmable Core Module processing cards from Z-World Engineering.

Caution
The **Simple Sequencer** does not incorporate safeguards for fail-safe operation. This board should not be used in any device in which its failure could cause injury, loss of life, or property damage. J.R. Kerr makes no warranties whatsoever regarding the performance, operation, or fitness of this board for any particular purpose.

J R KERR AUTOMATION ENGINEERING
www.jrkerr.com

2.0 Quick Start

Hardware setup
1. You will need the following items:
   - **Simple Sequencer** board
   - **Z232-485** Serial Port Converter board
   - One or more **NMC** control modules (**PIC-SERVO, PIC-STEP** or **PIC-I/O**)
   - One **NMC** communications cable per **NMC** controller (10 wire ribbon cable)
   - RS232 cable (DB9 male/DB9 female, straight)
   - Logic power supply (7.5 - 12vdc, 500 ma) with Waldom 2-pin header connector.
   - Windows **Simple Sequencer** programming software (available from *www.jrkerr.com*)
   - PC running Windows 95/98/NT
2. Plug the **Simple Sequencer** into the 40 pin socket on the **Z232-485**. The **Simple Sequencer** board should be overlapping the center of the **Z232-485** board.
3. Interconnect your **PIC-SERVO, PIC-STEP** or **PIC-I/O** boards with the **Z232-485** board with the 10 wire ribbon cables and set jumpers as shown. (See *Figure 1* below.)
4. On the **Z232-485** board, move jumpers JP3 and JP4 both to their 2-3 positions (jumpers towards the center of the board).
5. Connect the RS232 port of the **Z232-485** board to a free PC COM port.
6. Connect the logic supply to connector JP6 on the **Z232-485** board, but do not turn on.

Programming
1. Install the Simple Sequencer programming application software by copying all of the files (including the Examples folder) on the distribution disk to a single folder on your hard drive. (No formal installation process is necessary.)
2. Start the **Simple Sequencer** application software. Select the COM port when prompted.
3. Turn on the logic power supply. The **Simple Sequencer** program will display the message "Simple Sequencer Detected", followed by a message with the number of **NMC** modules detected. The module list on the left side of the display should list all modules you have interconnected.
4. Create the following simple program:
   a. Click on the "Master" module in the module list on the left.
   b. Select "NmcInit" from the Master Commands.
   c. Click on the "Add" button at the bottom of the Build a Command panel.
      This will add the first line to your program.
   d. Now select the "Print" command from the Master Commands list.
      Enter the text "Hi there." in the Print Characters edit box.
   e. Again, click on the "Add" button.
   f. Select "StopProgram" from the Master Commands list. Click on "Add".
      This completes your program.
5. Click the "Save" button underneath the Program Listing to save the program. (You can use the name "test.ssq".)
6. Click the "Load" button to download the program to the **Simple Sequencer**.
7. Before running your program, click on the "Terminal" button to monitor what gets printed by your program. (Move the terminal window to one side if necessary.)
8. Click the "Run" in the Program Execution panel. "Hi there." will appear in the terminal window.
9. Note that if you turn off the logic power and start up a generic terminal program (such as Hyperterminal with settings 19200 baud, 1 start bit, 1 stop bit, no handshaking), the text "Hi

there." should appear about 1 second after turning the logic power back on. Please read the rest
of this manual to discover all the other things you can do besides sending greetings.
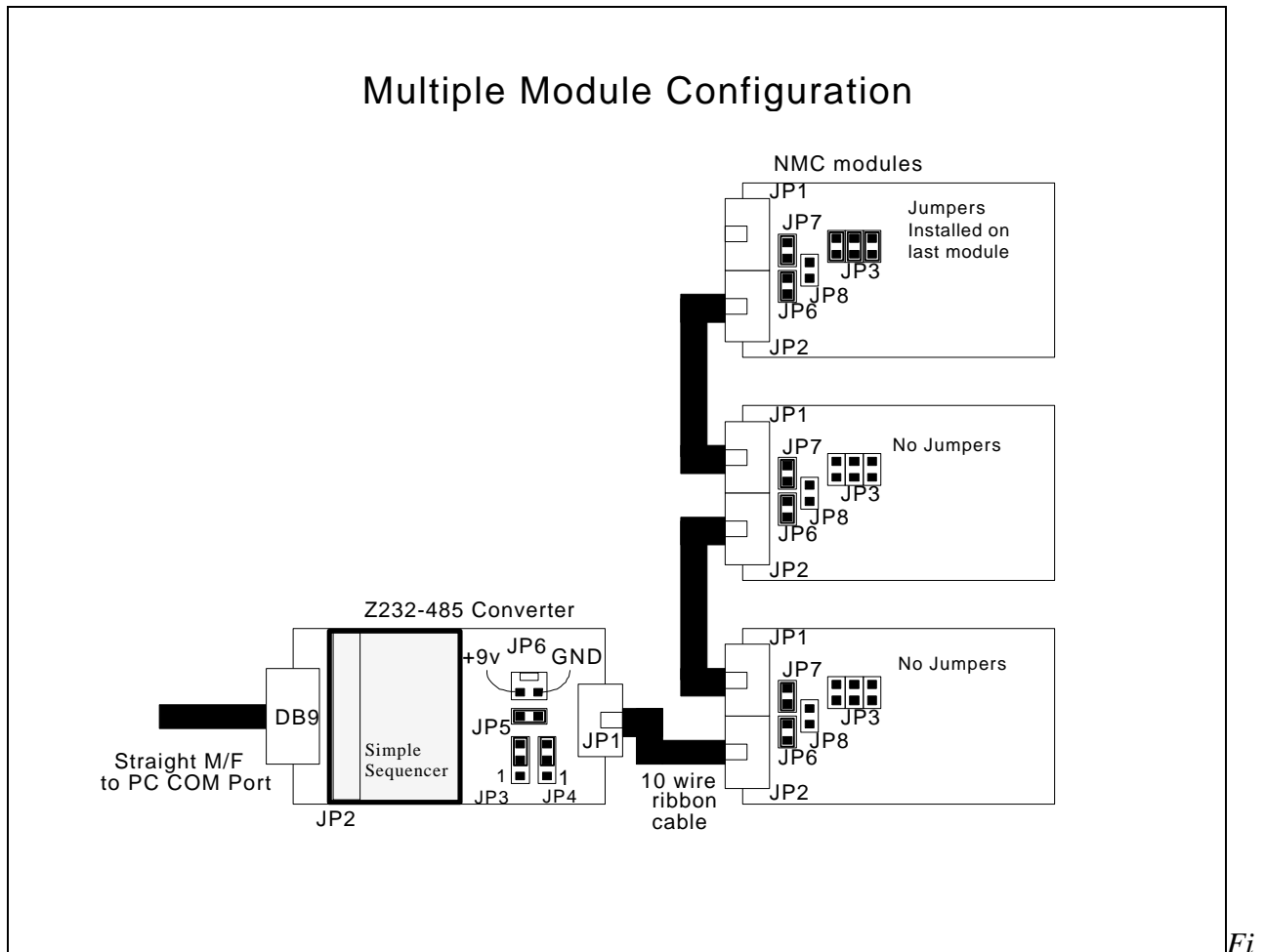

Multiple Module Configuration

Figure 1 - Interconnection diagram.

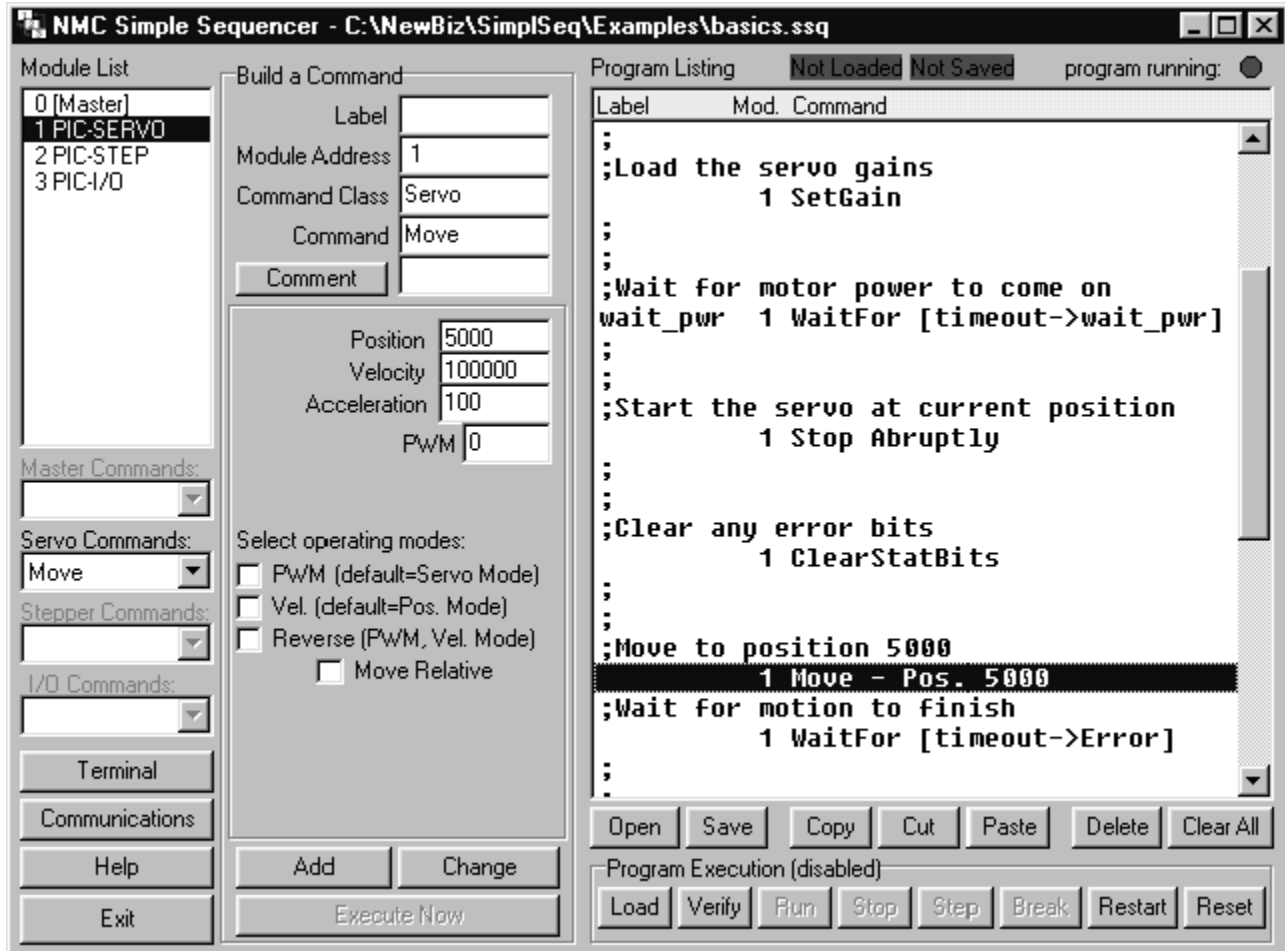## 3.0 *Simple Sequencer* Programming Screen

Figure 2 - **Simple Sequencer** Programming Screen.

Title Bar: Displays the file name of the program you are working on.

Module List: Displays the list of **NMC** modules. This module list is either detected automatically on power-up, loaded when you open an existing program, or created when you left-click on this list to edit it. The first module (Master) refers to the **Simple Sequencer** itself.

Clicking on a module will enable the appropriate Command List below. Double-clicking on a module will display its current status.

Command Lists: Master, Servo, Stepper or I/O Command lists list the specific commands for each type of module. The specific lists are enabled by clicking on a module in the Module List above.

Build a Command Panel: This is where you enter the specific data for a command. The top part of this panel displays information generic to all commands. The lower panel will display edit boxes, check boxes or other controls for entering data specific to a particular type of command.

> *Label:* Each command may optionally have a label. Enter a unique label here to be able to jump to this command using a *GoTo* or *Call* command.

> *Module Address:* Displays the address of the module associated with a command. This value is automatically filled in when you click on a module in the Module List or when you click on a command in the program listing.

Command Class: Displays the type of command. This value is automatically filled in when you click on a module in the Module List or when you click on a command in the program listing.

Command name: Displays command name. This value is automatically filled in when you click on a module in the Module List or when you click on a command in the program listing.

Comment Button: The Comment button inserts the text entered into the comment window next to it as a comment line in your program listing. When displayed in the program listing, comments are always preceded by a ';'. Comments have no effect on program execution, but are useful for documenting the operation of your program.

Add Button: The Add button adds a new command to your program listing. The new command is added *above* the highlighted program line.

Change Button: This button allows you to edit the parameters of an existing command or comment line.

Execute Now Button: This button will cause the command you have just created to be executed immediately by target module. Note that not all commands can be executed immediately.

Program Listing: The program listing displays the command sequence to be executed. The first part of a command listing is the label. Next is the address of the target module for that command. Next is brief command description. Clicking on a command will display the full details of that command in the "Build a Command" panel.

At the top of the Program Listing are several indicators: *Loaded/Not Loaded* indicates whether the current program has been downloaded into the **Simple Sequencer**, *Saved/Not Saved* indicates whether the program as been saved to disk or not, and the *program running* indicator light lets you know if the program is running (green) or stopped (red).

Open Button: Opens a **Simple Sequencer** program saved on disk.

Save button: Saves the current program to disk.

Copy Button: Copies selected program lines into a copy buffer. Several lines can be selected by either clicking on a line and dragging the cursor down over additional lines, or by clicking on one line and then shift-clicking on another line to select all the lines in-between.

Cut Button: Similar to the Copy button, except that it deletes the lines from your program after copying them to the copy buffer.

Paste Button: Pastes the program lines in the copy buffer into the location just *above* the highlighted program line. The program lines are left in the copy buffer and may be pasted again.

Delete Button: Deletes a single program line permanently.

Clear All Button: Clears your entire program. A prompt will appear to verify before your program is actually cleared.

Program Execution Panel: This panel contains controls for downloading and debugging your program. This panel is disabled if you are not working with your **Simple Sequencer** on-line.

Load Button: Downloads your program into the **Simple Sequencer** non-volatile memory.

Verify Button: Reads the **Simple Sequencer** memory and compares it to the current program listing. This is useful for verifying which program is actually loaded into the non-volatile memory.

Run Button: Begins or resumes execution of your program. If you want to make sure that you are running from the beginning, click on the Restart button first. As the program runs, the program line being executed will automatically be highlighted.

Stop Button: Halts execution of your program. Note that if a **PIC-SERVO** module is in the middle of executing a motion, **it will continue to execute that motion even though the program is stopped!**

Step Button: Causes just the current program line to be executed and the next line will be automatically be highlighted. Note that the current program line is the determined by the **Simple Sequencer's** internal program pointer.

Break Button: Sets a program breakpoint at the highlighted line, indicated by a '■' at the beginning of the program line. When the program encounters this line, it will stop the program before executing this line. Clicking on Run or Step will cause the program to continue executing past this line. Highlighting a line with a breakpoint and then clicking on Break will clear the break point. Only one breakpoint can be set.

Restart Button: Sets the **Simple Sequencer's** internal program pointer to the beginning of the program. Use this button to restart the program after a "StopProgram" command, or to restart in the middle of program execution.

Reset Button: Resets the **Simple Sequencer** and the connected modules to their power-up state. This button can also be used as a stop button to halt any motion in progress (although a hardwired emergency stop circuit should also be used).

Terminal Button: This button pops up a simple terminal window for sending and receiving characters to and from the **Simple Sequencer's** RS232 port..

Communications Button: Allows you to change the COM port you are using. If you are running off-line (that is, your **Simple Sequencer** is not connected to your PC), you can choose an "Offline" option rather than COM1, COM2, COM3 or COM4.

Help Button: Displays on-line help. (Requires the Acrobat Reader)

Exit Button: Terminates the Windows **Simple Sequencer** interface program. Note that terminating this program does not terminate the execution of the program which may be running on the **Simple Sequencer** itself.

4.0 Creating Programs

4.1 Creating a Module List
Before creating a program a program, you must first have a list of modules to work with. A module list can be created in one of three ways:

Working On-line with Modules Connected
If you have your PC's COM port connected to the **Simple Sequencer** with **NMC** modules attached (see *Figure 1*), the **Simple Sequencer** interface program will automatically detect how many modules are connected as well as the types of modules, and a module list will be created. In the list, the

Simple Sequencer itself appears as the Master module (address 0). The module furthest from the **Simple Sequencer** will be assigned address 1, with addresses increasing from there.

Creating a New Module List
If you are working off-line, or are writing a program for modules not currently connected, you can create a new module list by clicking with the right mouse button on the Module List. This will pop up a menu with options for clearing the list, adding modules, or using the modules automatically detected on-line. Modules must be added in the order in which they will be interconnected, with the first module being the furthest from the **Simple Sequencer**. When you save your program, this module list will be saved as part of the program.

Using a Program's Module List
When you open a previously written program, the module list stored as part of that program will be displayed in the Module List window.

If you edit the module list or open a program such that the module list does not match the modules connected on-line, a warning message will be displayed, and you will not be allowed to actually run your program.

CAUTION: If you do attempt to run a program stand-alone which does not have the appropriate modules connected and in the correct order, the results will be unpredictable and potentially hazardous. You should always be very careful when testing new programs, or when operating the **Simple Sequencer** after you have adjusted any of the cabling.

4.2 Creating Program Command Sequences
Creating program command sequences consist of the following three steps:
1. *Select a module to send a command.* Most commands will be sent to a **PIC-SERVO, PIC-STEP** or a **PIC-I/O** module. Commands controlling the program initialization or program flow, however, will be sent to the Master module (the **Simple Sequencer** itself.). Once a module is selected, the appropriate command list below will be enabled.
2. *Select a command from the command list.* Once a command is selected, places to enter command specific parameters will appear in the lower area of the "Build a Command" panel. Details of all of the commands are in *Section 6.0* (Programming Reference) below.
3. *Fill in the command specific data.* This data is entered into the "Build a Command" area. Note that some commands do not require any specific data to be entered. If you want your program to be able to jump to this specific command, enter a unique label at the top of this panel.

Once the command data and optional label have been entered click on the Add button to add the command to your program. The command will be added just *above* the highlighted program line. You can click on any line in the program to move the highlight, and thus control where the next new line gets added.

4.3 Editing Programs
Changing a Single Command
To change the parameters or label for a single command, first click on the command in the Program Listing, and the specific details for that command will appear in the "Build a Command" panel.

Change the parameters or label as required. Clicking on the Change button will update the command with the new parameters.

If you want to change a comment line, clicking on the comment in the Program Listing will display the comment text in the comment edit box. After modifying the comment, clicking on the change button below will update the comment.

Rearranging Program Lines
Rearranging program lines can be done with the Copy, Cut and Paste buttons. One or more program lines can be selected by clicking on a program line and then dragging the cursor over additional program lines. Alternately, clicking on one program line and then shift-clicking on another program line will select all lines in-between.

With program lines selected, clicking on the Copy button will copy those lines into an internal copy buffer, but will not delete them from the Program Listing. Clicking on the Cut button will also copy the lines, but they *will* be deleted from the Program Listing.

Once you have some lines in the copy buffer, you can paste them into a new location. Click on the program line just below where you want the lines inserted, and then click on Paste.

The Delete button will delete just a single highlighted line of your program without affecting the copy buffer. If you wish to delete several lines, you can select them and then use the Cut button instead. (Any subsequent Copy or Cut will flush any previous copied lines from the copy buffer.)

The Clear All button will delete everything in the program listing. You will be prompted to verify before this action is taken.

Saving and Opening Programs
Once you have created a program, you can save it using the Save button. A window will pop up with a place for you to enter the desired file name. The default extension for **Simple Sequencer** programs is '.ssq'. If you enter in a file name with no extension, an '.ssq' will be appended automatically. The module list will also be saved as part of the program.

When you save a program, the indicator above the Program Listing will indicate "Saved". Any change to the program or module list will change the indicator to "Not Saved". If you attempt to exit without saving your program, you will be prompted with an opportunity to save before exiting.

The Open button is used to open a previously written **Simple Sequencer** program. If you are working on-line and the module list for the program does not match the modules detected, the program will be opened, but you will not be allowed to run the program from this Windows application.

CAUTION: If you do attempt to run a program stand-alone which does not have the appropriate modules connected and in the correct order, the results will be unpredictable and potentially hazardous. You should always be very careful when testing new programs, or when operating the **Simple Sequencer** after you have adjusted any of the cabling.

4.4 Testing Individual Commands

It is often handy to be able to check the operation of a particular command or particular command parameters without having to enter the command into your program. Especially with motion control programs, you might want to test out positions, velocities or accelerations for your particular application. The Execute Now button allows you to execute a single command built up in the Build a Command panel without entering it into your program. The Execute Now option is only available with certain command for which immediate execution makes sense.

In conjunction with testing individual commands, it is also useful to be able to see the current state of limit switches, digital I/O lines, or motor positions. Double-clicking on a module in the module list will pop open a window displaying basic status information for the module. This window must be closed (with the '×' in the upper right corner) to resume operation. The data displayed is *not* dynamically updated while the window is open.

5.0 Running and Debugging Programs

Before testing your program, the **Simple Sequencer** must be powered up and connected to your PC. You can verify that the **Simple Sequencer** has been detected by looking at the Program Execution panel below the Program Listing. If the panel label reads: "Program Execution (disabled)", the **Simple Sequencer** has not been detected. If this is the case, make sure that the correct COM port has been selected, and then power-cycle the logic power for the **Simple Sequencer**. If everything is connected correctly, the message "Simple Sequencer Detected" will appear, followed by a message with the number of control modules found.

The RS232 port on the **Z232-485** converter board is used by the **Simple Sequencer** for both downloading and debugging, and for character I/O with an external computer or terminal. The Windows programming application gives you the ability to debug and perform terminal character I/O simultaneously.

5.1 Downloading and Verifying Programs

The Load button is used to download your program into the **Simple Sequencer's** non-volatile memory. This may take several seconds, depending on your program's size. (Programs download at about 1000 bytes per second.) When complete, the indicator above the Program Listing will switch to "Loaded".

If you wish to verify that the program in non-volatile memory matches the program listing, click on the verify button. If it matches, the Loaded/Not Loaded indicator will switch to "Loaded". This is most useful when you want to see if the loaded program matches one saved on disk.

The non-volatile memory on the **Simple Sequencer** is rated for 100,000 write cycles. This means that you can load programs at least 100,000 times. This should not present any problems for normal use, but you might want to avoid unnecessary clicking of the Load button.

5.2 Stand-alone Operation

Whenever you power-up the **Simple Sequencer**, it will send a single '#' character out its RS232 port and then wait for 1 second. If the Windows **Simple Sequencer** programming application is not connected and running, the **Simple Sequencer** will automatically begin execution of program in its non-volatile memory.

CAUTION: The *Simple Sequencer* cannot verify that *PIC-SERVO, PIC-STEP* and *PIC-I/O* modules are interconnected in the proper order. You should always verify the proper operation of your program using the Windows programming application before running your program stand-alone.

If your program contains any print statements or relies on character input to its RS232 port, you can connect its RS232 port to your PC and run any terminal program set up for 19200 baud, one start bit, one stop bit, no handshaking. When you first power-up, you will see a '#' character on the terminal, and the program will begin execution 1 second later.

You can change the RS232 baud rate in your *Simple Sequencer* program using the "ChangeBaud" command, but if you do so, you will not be able to run your program interactively through the *Simple Sequencer* programming application. Your program will only run stand-alone.

5.3 Running, Stopping and Stepping

With your program downloaded (and with the module list matching the modules detected) you can run your program interactively. If you want to run your program from the beginning, is always a good idea to click on the Restart button to insure that the program begins at the top. Clicking on Run will cause the program to begin execution.

The Windows programming application is constantly polling the *Simple Sequencer* at a rate of 4 times per second. As your program executes, the line being executed is automatically highlighted. Because many lines of your program may be executed between polling samples, the highlight may appear to skip over some lines. Most programs will spend the majority of their time waiting for some condition (*eg,* WaitFor *move done*) to be met. While the program is running, the program running indicator will be lit *green*.

When you click on the Stop button, the program will halt execution and the Program Listing highlight will land on the next instruction to execute. Clicking on Run will resume execution of the program. Program execution will also halt whenever a "StopProgram" command or a command with a breakpoint is encountered.

Clicking on Run at a breakpoint will resume program execution. Clicking on Run at a "StopProgram" command will have no effect. You must click on Restart to reset the program pointer to the beginning of the program. Clicking Run will then start execution from there.

Clicking on Step will cause just the next line of the program to execute. Normally, the next line to execute will be the highlighted line. If, however, you have repositioned the highlight by clicking on a different program line, clicking on Step or Run will still execute from the *Simple Sequencer's* internal program pointer which no longer matches the highlighted line.

If your program uses print commands or relies on RS232 character input, you will want to click on the Terminal button to pop open a simple terminal window. This window will display characters sent using Print commands, and can be used to send characters to the *Simple Sequencer* as well. Note that the RS232 port is being used simultaneously for character I/O and for monitoring the program's execution.

5.4 Using the Breakpoint

The breakpoint is used to automatically stop program execution for debugging purposes. While your program is stopped, you can click on a program line and then click on the break button. The first character in the line will be a '■' character to indicate the breakpoint has been set. When the program comes to a line with a breakpoint, it will halt execution without executing the breakpoint line. Clicking on Run or Step will resume execution starting with the breakpoint line.

To clear the breakpoint, click on the line with the breakpoint indicator, and then click on the Break button. The **Simple Sequencer** only supports one breakpoint. Breakpoints are cleared on power-up, and so they cannot be used in stand-alone operation.

5.5 Restart vs. Reset

The Restart button is used for setting the **Simple Sequencer's** internal program pointer to the beginning of the program after the program has stopped, and nothing else. The breakpoint is not affected, and the individual **NMC** modules may still be in some active state.

The Reset button, however, causes the **Simple Sequencer** to revert to its power-up state. When you click on Reset, you will again get the message "Simple Sequencer Detected", and the **NMC** modules will also be reset. In some instances, such as if the **Simple Sequencer** has had a communications fault with the **NMC** modules, the modules themselves may not be reset. In this case, the logic power should be turned off and on again.

As stated earlier, stopping the program execution on the **Simple Sequencer** does not necessarily stop any motion in progress. Therefore, the reset button can also be used as an stop button, causing all modules to reset. **However, it is always a good idea to have a separate emergency stop button connected directly to the motor power supply which will terminate any motion in spite of any communication faults.**

6.0 Programming Reference

Program Structure
For the most part, programs are simply lists of commands to be executed in order. A typical program might first initialize the **NMC** controllers (programs should always initialize the **NMC** controllers first), load servo gain parameters, start a motion, wait for the motion to finish, and then move on to the next motion, etc. .

The **Simple Sequencer**, however, offers a great deal more versatility by supporting program branching, subroutines, and conditional branching, and counters. Most commands associated with program flow control are Master commands, although conditional branching instructions (WaitFor and If-GoTo) which check module-specific data will also appear as Servo, Stepper or I/O commands. (The complete list of commands for each module type are listed in the sections below.)

The simplest branching is done with a "GoTo *label*" command which causes the program to jump to the program line with the matching label. The "GoTo" command is usually used in conjunction with conditional branching commands.

Conditional branching will cause the program to jump to a different line if some condition is met. For example the command "If-GoTo *label*" (using a condition of *Limit1 High*) will jump to *label* if

Limit Switch 1 is HIGH. (Note: a "GetStatus" command should be executed just prior to an "If-GoTo" statement to make sure the most current module status data is being tested.)

Conditional branching is also part of a "WaitFor" command. The "WaitFor" command essentially suspends program execution until a specified condition is met. "WaitFor" commands also have a timeout associated with them which specifies the maximum time to wait for the condition. If the maximum time elapses, the "WaitFor" command will then jump to the specified label. This conditional timeout feature allows your program to recover or take action should an expected condition fail to occur. ("WaitFor" commands can also jump to themselves to create an infinite timeout.)

The last form of conditional branching is associated with the "CountDown" command. The **Simple Sequencer** has four counters that can be set to initial values using the "SetCounter" command. The "CountDown" command will decrement a counter by 1, and then jump to the specified label if the counter does not equal zero. This allows you to repeatedly execute a set of program lines a specific number of times.

Another form of branching is the use of subroutines. There is no formal declaration for a subroutine, and subroutines can appear anywhere in your program listing. The first command in the subroutine should have a label, which is essentially the name of the subroutine. The last line of a subroutine should be a "Return" command. To call a subroutine, the "Call *label*" command is used, where *label* is the name of the subroutine. The after the lines of the subroutine are executed and the "Return" command is encountered, program control will continue execution at the line after the "Call" command.

Subroutines can be nested 10 levels deep. That is, one subroutine can call another subroutine, which can call another, and so on, up to 10 levels deep.

One last and very powerful feature of the **Simple Sequencer** is a special subroutine called the Monitor Subroutine. While the program in the middle of a "WaitFor" or "Delay" command, it is often desirable to be able to monitor various operating conditions and take corrective action should some error condition arise.   You can do this very simply by first writing a subroutine which reads modules' status ("GetStatus"), checks for any error conditions ("If-GoTo"), and returns if everything is OK. This subroutine can be given any name you want. To cause this Monitor Subroutine to be called automatically while in the middle of a "WaitFor" or "Delay", use the command "MonitorSub *label*" at the beginning of your program. Whenever your program comes to a "WaitFor" or "Delay" command, your monitor subroutine will automatically be called repeatedly in a loop until the "WaitFor" condition is met or times out, or until the "Delay" time expires. (Note that monitor subroutines themselves should not contain any "WaitFor" or "Delay" commands unless the Monitor Subroutine feature has been disabled.)

It is possible to have multiple monitor subroutines which are active at different times. Simply insert a new "MonitorSub *label*" command whenever you want to change monitor subroutines. (Use "MonitorSub NULL" to disable the use of any Monitor Subroutine.)

Communication External Computers

The **Simple Sequencer** uses the RS232 port both for downloading and for communication with external computers ore terminal devices. The "Print" command is used to transmit characters over the RS232 port.

The **Simple Sequencer** does not have a receive buffer, and therefore can only receive one character at a time. The "If-GoTo" or "WaitFor" commands are used to check if a character has been received or if the last received character is equal to a particular character. Note that each character sent to the **Simple Sequencer** will overwrite the previous character send. If any character is sent, the "Any Key" condition will become true and remain true until a "ClearKey command is executed.

Example Programs
The Windows **Simple Sequencer** programming application software includes many program examples illustrating the use of conditionals, counters, delays, "WaitFor" commands, and monitor subroutines. Please use these as templates as you start to write your own **Simple Sequencer** control programs. The following programs are included in the Examples folder:

> *basics.ssq* - Basic motor initialization and operation.
> *branching.ssq* - Examples using If-GoTo, WaitFor commands
> *counter.ssq* - Example using counters and loops.
> *subroutines.ssq* - Example of subroutines.
> *nesting.ssq* - Example of nested subroutines.
> *monitorsub.ssq* - Example using Monitor Subroutines.
> *console.ssq* - Simple RS232 operator interface.
> *homing5.ssq* - Example of homing a motor and resetting its position.
> > (**PIC-SERVO** firmware v.5 or higher)
> *homing4.ssq* - Example of homing a motor and resetting its position.
> > (**PIC-SERVO** firmware v.3 or v.4)

6.1 Master Commands

GoTo	Unconditional jump to a label
Call	Call a subroutine with the specified label
Return	Returns to main program after a subroutine call
If-Goto	Branch on conditions particular to the Master module
StopProgram	Halt program execution
Print	Print characters out the RS232 port
SetTimer	Set the **Simple Sequencer's** internal timer
GetStatus	Update the current timer, counter and RS232 key input status from the **Simple Sequencer**
MonitorSub	Define the label for the active Monitor Subroutine
Delay	Delay program execution for a specified time
WaitFor	Wait for RS232 key input.
ClearKey	Clear RS232 key input
SetCounter	Set the initial value for one of the four counters
CountDown	Decrement one of the four counters & loop to a label if not zero
NmcInit	Initialize the NMC controller network.
Reset	Reset the **Simple Sequencer** to its power-up state
ChangeBaud	Change the RS232 port baud rate

GoTo

> Parameters
>> *label*   Program label to jump to

GoTo unconditionally jumps to the program instruction with the label *label*.

Call

> Parameters
>> *label*   Program label of subroutine

Call unconditionally jumps to the subroutine instruction with the label *label*. When the program comes to the next "Return" command, program execution will continue at the statement after the "Call" command. One subroutines can call another subroutine, up to 10 levels deep.

Return

> Parameters
>> *none*

Return is used at the end of a subroutine to return program execution at the line after the corresponding "Call" command. A subroutine can have multiple "Return" commands depending on the flow of your logic. You should make sure that no matter what happens, your subroutine will eventually come to a "Return" command. If a "Return" command is encountered without a corresponding "Call" command, the results will be unpredictable!

If-GoTo (for Master)

> Parameters
>> *label*   Program label to jump to
>> *jump condition (one of the following):*
>>> Key input = specific character (from RS232 port)
>>> Any key input (condition remains true until a ClearKey
>>>> command is executed)
>>> Timer timeout
>>> Counter N = *count value* (N = 1, 2, 3 or 4, count value = 0 to 255)
>>> Number of modules connected = N

If-GoTo jumps to the program instruction with the label *label* if the selected jump condition is true. Note: before executing this command, you should always first execute a "GetStatus" command for the Master module to insure that you are examining the proper and current status data. If you have several "If-GoTo" statements *for the Master module*, they may be proceeded by a single "GetStatus" command. If, however, there are any other commands interspersed between the "If-GoTo" commands, you must place a "GetStatus" command before each "If-GoTo" command. For example:

OK:
```
;For Master:
     0 GetStatus
     0 If-GoTo (key = 'A') SUB_A
     0 If-Goto (key = 'B') SUB_B
     0 If-GoTo (key = 'C') SUB_C
```
NOT OK:
```
;For Master:
     0 GetStatus
     0 If-GoTo (key = 'A') SUB_A
;Intervening Servo command
     1 Stop - Motor Off
     0 If-Goto (key = 'B') SUB_B
     0 If-GoTo (key = 'C') SUB_C
```

StopProgram

Parameters
> *none*

StopProgram halts the program execution. It is typically used only when an error condition occurs, or when the program does not need to execute again until the system is powered up again.

Print

Parameters
> *chars*   1-16 characters to be printed out the RS232 port
> (with optional line-feed or carriage return chars.)

Print sends the specified characters out over the Z232-485 board's RS232 port. This can be used for communicating with external computers or for printing text to a serial terminal or serial LCD display. Characters must be ASCII printable characters.

SetTimer

Parameters
> *time*   Timer value in units of $1/100^{th}$ seconds

SetTimer sets the internal timer of the **Simple Sequencer** but does not delay program execution Detecting when the timer times out can be done using the "If-GoTo" command for the Master module. Note that *any* "WaitFor" or "Delay" command will reset the internal timer to a new value.

GetStatus (for Master)

Parameters
> *none*

GetStatus loads the status data for the Master module into the **Simple Sequencer's** internal status buffer. The "If-GoTo" command then operates on current data in this status buffer. Note that each module type has its own variant of the If-GoTo command which expects to find corresponding type of status data in the internal status buffer. Therefore, a "GetStatus" command for a module should always precede an "If-Goto" command for that module.

MonitorSub

      Parameters

            label   Subroutine label for the Monitor Subroutine

                    ("NULL" may be used for no Monitor Subroutine)

MonitorSub tells the **Simple Sequencer** the name of the subroutine it should use as the Monitor Subroutine. The Monitor Subroutine will then be automatically called in a loop whenever a "Delay" or a "WaitFor" command is executed. The Monitor Subroutine is typically used for detecting error conditions and taking corrective action. The "MonitorSub" command can be used to change back and forth between different Monitor Subroutines, or to define a "NULL" Monitor Subroutine. *Note:* Monitor Subroutines should not contain any "Delay" or "WaitFor" statements unless the Monitor Subroutine feature has been disabled first (using MontiorSub NULL).

Delay

      Parameters

            time   Delay time in $1/100^{th}$ seconds

The Delay command delays program execution for the specified amount of time.

WaitFor (for Master)

      Parameters

            condition (one of the following)

                    Key input = specific character (from RS232 port)

                    Any key input (condition remains true until a ClearKey

                              command is executed)

            timeout   Timeout value in $1/100^{th}$ seconds

            label   Program label to jump to on timeout

WaitFor halts program execution until the specified condition is met. If the condition is not met before the timeout period expired, the program will jump to *label*. To create an infinite timeout time, you can give this command a program label and have it jump to itself on timeout. Note that the "WaitFor" command automatically polls the corresponding module to get its current status data (no GetStatus command is needed). If a timeout value of 0 is used, the condition will be checked once, and the program will jump immediately to *label* if the condition is not true, or continue to the next command if the condition is true.

ClearKey

      Parameters

            none

When a character is received from the RS232 port (via terminal key press or external computer input), it remains in place. "ClearKey" clears the last character input from the RS232 port. "ClearKey" should be used before waiting for character input from a terminal or external computer.

SetCounter

> Parameters
>> *counter*   Counter number (1, 2, 3 or 4)
>> count   Initial counter value (0 - 65535)

SetCounter sets the initial value of one of the four counters. Program loops can then be created using the "CountDown" command. Note that when testing the counter value using the "If-GoTo" command, only counter values between 0 and 255 can be compared.

CountDown

> Parameters
>> *counter*   Counter number (1, 2, 3 or 4)
>> *label*   Label to jump to if counter is not equal to zero

CountDown decrements the value of a counter by 1. After decrementing, if the counter is equal to zero, the program will jump to *label*. This is a convenient structure for creating program loops which execute a fixed number of times. If you want to decrement a counter without looping anywhere if not zero, simple give the following program line a label and jump to that label if not zero. This makes "CountDown" have no effect on the flow of the program.

NmcInit

> Parameters
>> *none*

NmcInit initializes the network of **NMC** controllers, determines the number of modules connected and assigns them their addresses (starting with 1 for the module furthest from the **Simple Sequencer**). The "If-GoTo" command for the Master module can be used to verify that the number of modules detected matches that required by the program. "NmcInit" is typically used as the first line of every program.

Reset

> Parameters
>> *none*

The Reset command resets the **Simple Sequencer** to its power-up state. Note that this does not necessarily reset the network of NMC controllers unless an "NmcInit" command appears at the beginning of the program. When a "Reset" command is encountered, there will be a 1 second delay before program execution automatically restarts.

ChangeBaud

> Parameters
>> *baud rate*   1200, 2400, 9600 or 19200

ChangeBaud changes the baud rate of the RS232 port for communication with a terminal or external computer. (The default baud rate is 19200.) Programs which contain the "ChangeBaud" command cannot be run interactively because the **Simple Sequencer** programming application needs to communicate at the default baud rate - you will only be able to run your program stand-alone

6.2 Servo Commands

Move	Move in position, velocity or PWM control modes
Stop	Stop a motor smoothly, abruptly, or with the servo disabled
SetGain	Set the servo parameters for motion control
ResetPosition	Reset the position motor position counter
StartHoming	Start searching for homing switches
GetStatus	Update the current motor status data
ClearStatBits	Clear error status bits for a motor
If-GoTo	Branch on conditions particular to a **PIC-SERVO** module
WaitFor	Wait for a **PIC-SERVO** operating condition (move done, etc.)

Move

Parameters

position   Goal position (+/- 2,000,000,000 counts)
(use zero if in velocity or PWM mode)

velocity   Goal/Slew velocity (0 to 65,535)
(use 0 if in PWM mode)

acceleration   Acceleration/Deceleration value (0 to 65,535)
(use 0 if in PWM mode)

PWM   PWM output value (0 to 255)
(use 0 if in position or velocity mode)

PWM option   Selects raw PWM output mode

Vel. Option   Selects velocity mode option

Rev. Option   Makes PWM or Velocity value negative
(only valid for PWM or velocity modes)

Move Relative Moves relative to current position
(only valid for position mode)

The Move command is used for almost all motion commands in either position, velocity or raw PWM modes. In position mode, the *position*, *velocity* and *acceleration* are used to specify a trapezoidal profile motion. In velocity mode, the *velocity* and *acceleration* are used to specify an acceleration limited velocity command, with the *reverse option* used to specify a negative velocity (*position* is ignored). In PWM mode, the *PWM* value sends a raw drive signal to the motor (0 = 0%, 255 = 100%), with the *reverse option* used to specify a negative drive output.

The move relative option is only valid in position mode, and it specifies that the goal position is relative to the current position. The move relative option is only available for **PIC-SERVO** firmware v.5

The motor should be stopped initially before commanding a position mode move, and the move should be completed before issuing another position mode command. However, velocity or PWM mode commands can be issued at any time, even if in the middle of a position mode move.

Motor motion will start immediately when the "Move" command is executed, and program execution will proceed to the next command without waiting for the motion to complete. This enables you to

start the motion of several motors, and then use the "WaitFor" command for each motion to complete.

Positions are in units of encoder counts, velocities in units of encoder counts per servo tick, and accelerations in units of encoder counts per servo tick per servo tick. To increase the integer resolution of velocities and accelerations, they are actually specified as the calculated values (in terms of encoder counts and servo ticks) multiplied by 65,536. The default servo tick is about 1/2000th second. Please refer to the **PIC-SERVO** chipset documentation for more details on specifying motion control parameters.

Stop
Parameters

enable amplifier	Option to enable/disable the amplifier
motor off option	Drive signal to motor is disabled
stop abruptly option	Motor stops immediately, and holds its position
stop smoothly option	Motor decelerates smoothly and holds its position

Stop is used both for stopping the motor when moving and for initially turning on the PID servo. The *enable amplifier* option is also used for setting an I/O bit to enable the servo amplifier. On initialization, the "Stop" command with *enable amplifier* selected and *stop abruptly* selected will cause the motor to start servoing to its current position.

SetGain
Parameters

Kp	Proportional gain (0 to 32,767)
Kd	Derivative gain (0 to 32,767)
Ki	Integral gain (0 to 32,767)
IL	Integration limit (0 to 32,767)
OL	Output limit (0 to 255)
CL	Current limit (0 to 255)
EL	Position error limit (0 to 16,383)
SR	Servo rate divisor (1 - 255)
DC	Deadband compensation (0 - 255)

SetGain sets the operating parameters for the P.I.D. servo control. The parameters of most interest are *Kp*, *Kd* and *Ki*. *Kp* controls the stiffness of the servo regulating how tightly the actual position will match the goal position. *Kd* controls the damping of the system, making it more sluggish but more stable. *Ki* controls the steady state error of the motor when its position is allowed to settle. (Note that *IL* must be set to a non-zero value, typically 500 - 2000, if a non-zero value of *Ki* is used.)

A "SetGain" command should be used before enabling the motor servo. The "SetGain" command can be used at any time to change servo control parameters for different operating conditions.

Selecting the best values for the servo control parameters will take a bit of experimentation, but the default values in the **Simple Sequencer** programming utility should give you a good starting point. Please refer to the **PIC-SERVO** chipset data sheet for a more complete description of the servo control parameters.

ResetPosition

> Parameters
>> *relative to home*   Option to reset relative to the home position

ResetPosition resets the motor position counter to zero. If the *relative to home* option is selected, the counter will be reset to a counter value relative to the home position captured by using the "StartHoming" command. The *relative to home* option is only available for **PIC-SERVO** v.5 firmware or later.

StartHoming

> Parameters
>> *homing condition (one or more of the following)*
>>> Change of limit switch 1
>>> Change of limit switch 2
>>> Change of encoder index signal
>>> Position error condition
>>> Current limit condition
>> *Autostop option*   Stops smoothly, abruptly, or with motor off on home

StartHoming sets the **PIC-SERVO** into a mode where it looks for one or more homing conditions to become true. When one of the homing conditions is detected, the motor position is stored in an internal register, and if specified, the motor will stop in the selected manner. Note that "StartHoming" does not actually start any motor motion - it is typically used just before a "Move" command which drives the motor into one of the limit switches. Very often, one homing command using the limit switches will be followed by another homing command using the encoder index for a more accurate position. Please examine the example programs *homing4.ssq* or *homing5.ssq* for more details.

GetStatus (for **PIC-SERVO**)

> Parameters
>> *none*

GetStatus loads the status data for a **PIC-SERVO** module into the **Simple Sequencer's** internal status buffer. The "If-GoTo" command then operates on current data in this status buffer. Note that each module type has its own variant of the If-GoTo command which expects to find corresponding type of status data in the internal status buffer. Therefore, a "GetStatus" command for a module should always precede an "If-Goto" command for that module.

ClearStatBits

> Parameters
>> *none*

ClearStatBits clears various "sticky" error condition bits for a **PIC-SERVO** module including the position error bit. "ClearStatBits" should be used after motor servos are turned on. A position error condition can be monitored using the "If-GoTo" command or the "WaitFor" command, and appropriate action can be taken.

If-GoTo (for *PIC-SERVO*)

    <u>Parameters</u>

        label   program label to jump to

        jump condition (one of the following)

            move done / not done

            power on / not on

            overcurrent / no overcurrent

            position error / no position error

            Limit switch 1 high / low

            Limit switch 2 high / low

            Homing in progress / complete

If-GoTo will jump to the program label *label* if the selected condition is true. Note: before executing this command, you should always first execute a "GetStatus" command for the same **PIC-SERVO** module to insure that you are examining the proper status data.

WaitFor (for *PIC-SERVO*)

    <u>Parameters</u>

        condition (one of the following)

            move done / not done

            power on / not on

            overcurrent / no overcurrent

            position error / no position error

            Limit switch 1 high / low

            Limit switch 2 high / low

            Homing in progress / homing complete

        timeout   Timeout value in $1/100^{th}$ seconds

        label   Program label to jump to on timeout

WaitFor halts program execution until the specified condition is met. If the condition is not met before the timeout period expired, the program will jump to *label*. To create an infinite timeout time, you can give this statement a program label and have it jump to itself on timeout. Note that the "WaitFor" command automatically polls the corresponding module to get its current status data (no GetStatus command is needed). If a timeout value of 0 is used, the condition will be checked once, and the program will jump immediately to *label* if the condition is not true, or continue to the next command if the condition is true.

6.2 Stepper Commands

Move	Move in position or velocity control modes
Stop	Stop a motor smoothly, abruptly, or with the amplifier disabled
SetParam	Set motion parameters
SetOutputs	Set output bits and amplifier control bits
ResetPosition	Reset the motor position counter
StartHoming	Start searching for homing switches
GetStatus	Update the current motor status data
If-GoTo	Branch on conditions particular to a **PIC-TEP** module
WaitFor	Wait for a **PIC-STEP** operating condition (move done, etc.)

Move

Parameters

position	Goal position (+/- 2,000,000,000 counts)
	(use zero if in velocity mode)
speed	Goal/Slew speed (1 to 250)
accel. time	Acceleration/Deceleration time (1 to 255)
Vel. Option	Selects velocity mode option
Rev. Option	Makes velocity value negative
	(only valid for velocity mode)

The Move command is used for almost all motion commands in either position, velocity or raw PWM modes. In position mode, the *position*, *speed* and *acceleration time* are used to specify a trapezoidal profile motion. In velocity mode, the *velocity* and *acceleration* are used to specify an acceleration limited velocity command, with the *reverse option* used to specify a negative velocity (*position* is ignored).

The motor should be stopped initially before commanding a position mode move, and the move should be completed before issuing another position mode command. However, velocity mode commands can be issued at any time, even if in the middle of a position mode move.

Motor motion will start immediately when the "Move" command is executed, and program execution will proceed to the next command without waiting for the motion to complete. This enables you to start the motion of several motors, and then use the "WaitFor" command for each motion to complete.

Positions are given as the number of steps. Speeds are in increments of 25 steps per second (in 1x speed mode). The acceleration time, multiplied by 0.25 milliseconds, is the amount of time spent at each integer speed value before accelerating to the next highest speed. Please refer to the **PIC-STEP** documentation for more details on specifying motion control parameters.

Stop

Parameters

enable amplifier	Option to enable/disable the amplifier
stop abruptly option	Motor stops immediately, and holds its position
stop smoothly option	Motor decelerates smoothly and holds its position

Stop is used both for stopping the motor when moving and for initially enabling the amplifier using the *enable amplifier* option. On initialization, the "Stop" command with *enable amplifier* selected and *stop abruptly* selected will cause the motor to start holding to its current position.

SetParam

Parameters

min. speed	Minimum speed (1-250)
run current	Running current (0-255)
hold current	Holding Current (0-255
therm. limit	Thermal limit (0-255)
ignore limits	Ignore limit switches for auto-stop
ignore e-stop	Ignore E-Stop input
speed modes	Stepping speed options (1x, 2x, 4x or 8x)
off on e-stop	Turn motor off on E-Stop or Limit switch

SetParam sets the operating parameters for the stepper controller. The minimum speed is the lowest running speed used during acceleration or deceleration. The running and holding current dictate how much current is supplied to the motor while moving or stationary. A value of 255 produces the maximum current, and a value of zero produces no current. Because the **PIC-STEP**'s amplifier can be overdriven by approximately 20%, <u>the running or holding current should not be set to over 200 for more than a few seconds.</u> The thermal limit can be used if an NTC (negative temperature coefficient) thermistor is attached to the motor and connected to the **PIC-STEP** board. (Use a thermal limit of zero if a thermistor is not used.)

The *ignore limits* option disables the **PIC-STEP**'s feature which causes the motor to stop automatically when a limit switch is hit. The *ignore e-stop* option disables the feature which automatically stops the motor when the E-Stop input is active. The *off on e-stop* option causes the amplifier to be disabled if the motor is ever stopped automatically by limit switch or E-Stop inputs.

The *speed modes* allow you to select the operating speed of the step timer. The 1x speed is suitable for most motors, but the higher speeds may be used when driving high resolution motors, or when using an external microstepping driver. In the higher speed modes, the step rate is multiplied by 2x, 4x or 8x.

A "SetParam" command should be used before enabling the amplifier. The "SetParam" command can be used at any time to change motion control parameters for different operating conditions. Please refer to the **PIC-STEP** data sheet for a more complete description of the servo control parameters.

SetOutputs

Parameters

output1	Unused output bit
half step	Selects half-stepping mode (*default*)
slow decay	Selects slow-decay mode (*default*)
run mode	Selects running mode (*default*)
output5	Aux. output bit

SetOutputs is used to set output bits which will govern the operation of the amplifier. In most cases the default settings should be used. A SetOutput command should be issued before the amplifier is enabled. Please refer to the **PIC-STEP** motor control board data sheet for complete details.

ResetPosition
> <u>Parameters</u>
> > *none*

ResetPosition resets the motor position counter to zero

StartHoming
> <u>Parameters</u>
> > *homing condition (one or more of the following)*
> > > Change of limit switch 1
> > > Change of limit switch 2
> > > Change of homing switch
> > *Autostop option*   Stops smoothly or abruptly on home

StartHoming sets the **PIC-STEP** into a mode where it looks for one or more homing conditions to become true. When one of the homing conditions is detected, the motor position is stored in an internal register, and if specified, the motor will stop in the selected manner. Note that "StartHoming" does not actually start any motor motion - it is typically used just before a "Move" command which drives the motor into one of the limit switches.

GetStatus (for *PIC-STEP*)
> <u>Parameters</u>
> > *none*

GetStatus loads the status data for a **PIC-STEP** module into the **Simple Sequencer's** internal status buffer. The "If-GoTo" command then operates on current data in this status buffer. Note that each module type has its own variant of the If-GoTo command which expects to find corresponding type of status data in the internal status buffer. Therefore, a "GetStatus" command for a module should always precede an "If-Goto" command for that module.

If-GoTo (for *PIC-STEP*)

    <u>Parameters</u>

        label   program label to jump to

        jump condition (one of the following)

            moving / not moving

            power on / not on

            amp enabled / disabled

            at speed / not at speed

            homing in progress / complete

            limit switch 1 high / low

            limit switch 2 high / low

            homing switch high / low

            e-stop high / low

            aux. input 1 high / low

            aux. input 2 high / low

If-GoTo will jump to the program label *label* if the selected condition is true. Note: before executing this command, you should always first execute a "GetStatus" command for the same **PIC-STEP** module to insure that you are examining the proper status data.

WaitFor (for *PIC-STEP*)

    <u>Parameters</u>

        condition (one of the following)

            moving / not moving

            power on / not on

            amp enabled / disabled

            at speed / not at speed

            homing in progress / complete

            limit switch 1 high / low

            limit switch 2 high / low

            homing switch high / low

            e-stop high / low

            aux. input 1 high / low

            aux. input 2 high / low

        timeout   Timeout value in $1/100^{th}$ seconds

        label   Program label to jump to on timeout

WaitFor halts program execution until the specified condition is met. If the condition is not met before the timeout period expired, the program will jump to *label*. To create an infinite timeout time, you can give this statement a program label and have it jump to itself on timeout. Note that the "WaitFor" command automatically polls the corresponding module to get its current status data (no GetStatus command is needed). If a timeout value of 0 is used, the condition will be checked once, and the program will jump immediately to *label* if the condition is not true, or continue to the next command if the condition is true.

6.3 I/O Commands

MakeOutputs	Specify which I/O bits are outputs
SetOutputs	Set or Clear specific output bits
SetPWMs	Set the two PWM output values
GetStatus	Update the current **PIC-I/O** status data
If-GoTo	Branch on conditions particular to a **PIC-I/O** module
WaitFor	Wait for a PIC-I/O operating condition (input bit set or clear, etc.)

MakeOutputs

Parameters
individual bit selections Selects bits to be outputs (otherwise, they are inputs)

MakeOutputs allows you to select which of the 12 I/O bits on the **PIC-I/O** will be used as outputs. (On powerup, all bits are inputs). Bits can be switched back and forth between input and output for applications requiring bi-directional I/O, but care should be taken to make sure that a bit defined as an output will never be connected to another output or other low impedance connection - *this will damage the I/O port*.

SetOutputs

Parameters
individual bit selections Selects bits to change
set/clear option   Option to set or clear selected bits

SetOutputs sets or clears one or more output bits at a time. Bits not selected are unaffected. If a bit which is an input is selected to be set or cleared, no change will be seen at the corresponding I/O pin. However, setting or clearing an input bit prior to making it an output will guarantee its initial state when it becomes an output.

SetPWMs

Parameters
PWM1   PWM1 output value (0 - 255)
PWM2   PWM2 output value (0 - 255)

SetPWMs sets the values for the two high current PWM output drivers. A value of 0 corresponds to a 0% output signal, a value of 255 corresponds to a 100% output signal.

GetStatus (for **PIC-I/O**)

Parameters
none

GetStatus loads the status data for a **PIC-I/O** module into the **Simple Sequencer's** internal status buffer. The "If-GoTo" command then operates on current data in this status buffer. Note that each module type has its own variant of the If-GoTo command which expects to find corresponding type of status data in the internal status buffer. Therefore, a "GetStatus" command for a module should always precede an "If-Goto" command for that module.

If-GoTo (for *PIC-I/O*)
> <u>Parameters</u>
>> *label*   program label to jump to
>> *jump condition (one of the following)*
>>> bits 1 - 12 set or clear

If-GoTo will jump to the program label if the selected bit is set or cleared. Both the values of input bits and output bits can be used as a jump condition. Note: before executing this command, you should always first execute a "GetStatus" command for the same *PIC-I/O* module to insure that you are examining the proper status data.

WaitFor (for *PIC-I/O*)
> <u>Parameters</u>
>> *condition (one of the following)*
>>> bits 1 - 12 set / clear
>> *timeout*   Timeout value in $1/100^{th}$ seconds
>> *label*   Program label to jump to on timeout

WaitFor halts program execution until the selected bit is set or cleared. If the condition is not met before the timeout period expired, the program will jump to *label*. To create an infinite timeout time, you can give this statement a program label and have it jump to itself on timeout. Note that the "WaitFor" command automatically polls the corresponding module to get its current status data (no GetStatus command is needed). If a timeout value of 0 is used, the condition will be checked once, and the program will jump immediately to *label* if the condition is not true, or continue to the next command if the condition is true.

7.0 Other Documentation

The following additional documents will be useful for developing your application with the **Simple Sequencer**. All of these documents, as well as test software, can be downloaded from the **J R KERR** web site "*www.jrkerr.com*":

PIC-SERVO Motor Control Board manual	*PSBOARD.PDF*
PIC-SERVO Chipset data sheet	*PSDATA.PDF*
PIC-STEP Motor Control Board manual	*PSTEP_BD.PDF*
PIC-STEP Chip data sheet	*PICSTEP.PDF*
PIC-I/O Board manual	*PICIO.PDF*
Z232-485 Board manual	*Z232485.PDF*

For technical support regarding **NMC** controller products, please send e-mail to:

techsupport@jrkerr.com