Soft Servo
SYSTEMS, INC

ServoWorks CNC Macro Programming Manual

Revision 1.94

Warning

The product described herein has the potential – through misuse, inattention, or lack of understanding – to create conditions that could result in personal injury, damage to equipment, or damage to the product(s) described herein. Machinery in motion and high-power, high-current servo drives can be dangerous; potentially hazardous situations such as runaway motors could result in death; burning or other serious personal injury to personnel; damage to equipment or machinery; or economic loss if procedures aren't followed properly. Soft Servo Systems, Inc. assumes no liability for any personal injury, property damage, losses or claims arising from misapplication of its products. In no event shall Soft Servo Systems, Inc. or its suppliers be liable to you or any other person for any incidental collateral, special or consequential damages to machines or products, including without limitation, property damage, damages for loss of profits, loss of customers, loss of goodwill, work stoppage, data loss, computer failure or malfunction claims by any party other than you, or any and all similar damages or loss even if Soft Servo Systems, Inc., its suppliers, or its agent has been advised of the possibility of such damages.

It is therefore necessary for any and all personnel involved in the installation, maintenance, or use of these products to thoroughly read this manual and related manuals and understand their contents. Soft Servo Systems, Inc. stands ready to answer any questions or clarify any confusion related to these products in as timely a manner as possible.

The selection and application of Soft Servo Systems, Inc.'s products remain the responsibility of the equipment designer or end user. Soft Servo Systems, Inc. accepts no responsibility for the way its controls are incorporated into a machine tool or factory automation setting. Any documentation and warnings provided by Soft Servo Systems, Inc. must be promptly provided to any end users.

This document is based on information that was available at the time of publication. All efforts have been made to ensure that this document is accurate and complete. However, due to the widely varying uses of this product, and the variety of software and hardware configurations possible in connection with these uses, the information contained in this manual does not purport to cover every possible situation, contingency or variation in hardware or software configuration that could possibly arise in connection with the installation, maintenance, and use of the products described herein. Soft Servo Systems, Inc. assumes no obligations of notice to holders of this document with respect to changes subsequently made. Under no circumstances will Soft Servo Systems, Inc. be liable for any damages or injuries resulting from any defect or omission in this manual.

Soft Servo Systems, Inc. makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. NO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS OF PURPOSE SHALL APPLY.

Important Notice

The information contained in this manual is intended to be used only for the purposes agreed upon in the related contract with Soft Servo Systems, Inc. All material contained herein is subject to restricted rights and restrictions set forth in the contract between the parties.

These manuals contain confidential and proprietary information that is not to be shared with, nor distributed to, third parties by any means without the prior express, written permission of Soft Servo Systems, Inc. No materials contained herein are to be duplicated or reproduced in whole or in part without the express, written permission of Soft Servo Systems, Inc.

Although every effort and precaution has been taken in preparing this manual, the information contained herein is subject to change without notice. This is because Soft Servo Systems, Inc. is constantly striving to improve its products. Soft Servo Systems, Inc. assumes no responsibility for errors or omissions.

⚠ CAUTION

Due to the preread/block buffering aspect of macro statement execution, it is possible that macro statements using system variables may be evaluated, parsed, and delivered to the block buffer BEFORE the previous lines have been executed, and before the relevant system parameters have been updated. Therefore, the evaluation of a macro statement containing a system variable may not evaluate as expected.

IF YOU DO NOT PROGRAM IN BLOCK DELAYS BEFORE STATEMENTS INVOLVING SYSTEM VARIABLES, YOU WILL GET UNEXPECTED RESULTS IN THE EXECUTION OF YOUR MACRO PROGRAMS.

For more information, see *Section 7.1: Preread Function/Block Buffering – Problems and Workaround*.

Table of Contents

List of Tables

List of Figures

Chapter 1: Overview

1.1 What Is A Macro Program?

A macro is one instruction that is included in a part program and that represents a sequence of simpler instructions, known as a macro (either a macro subroutine or a macro subprogram, depending on where the macro code is located and how it is called). In a way, macros are like simple programs or batch files. ServoWorks CNC systems support sophisticated custom macros that allow you to use variables and flow control structures such as loops. In this way, you can use program logic to eliminate redundant programming of machine functions, such as milling multiple pockets.

Macros significantly simplify part programming and reduce part programming time by allowing reuse of programming pieces such as user-defined canned cycles. The extensive macro programming capabilities available with ServoWorks CNC systems will save time and reduce operator errors.

Macros are written in the ServoWorks macro programming language, which is similar to the BASIC programming language. You will find it easy to learn to program macros for ServoWorks CNC systems.

Macros can be used as subroutines, contained in the same file as the main program (and called using the G65 simple macro call). Or, macros can be used as separate macro subprograms, either residing in the same folder as the main program (and called using an M98 subprogram call), or residing anywhere, even on a network (using custom G, M, S or T macro calls).

1.2 Differences Between Simple (Non-Macro) Subprograms And Macros

Using a simple NC subprogram (without the macro programming option) for part programming allows reuse of code for repeating the same exact machining operation, but it is limited in terms of reusability and it is not very powerful.

Using macro subroutines (or macro subprograms) is very similar to calling a simple NC subprogram from a main part program, except with the added advantage that you can use variables, control structures and mathematical and logical operations in macros. This means that macros are much more powerful, more versatile and more reusable than simple NC subprograms. For instance, you can have one macro for a bolt hole, and just change the variables (arguments) you assign when you call the macro (with macro instructions). If you were using a simple (non-macro) subprogram for programming a bolt hole, you would need a different subprogram for each size bolt hole.

The format of a macro subroutine is the same format as a simple NC subprogram:

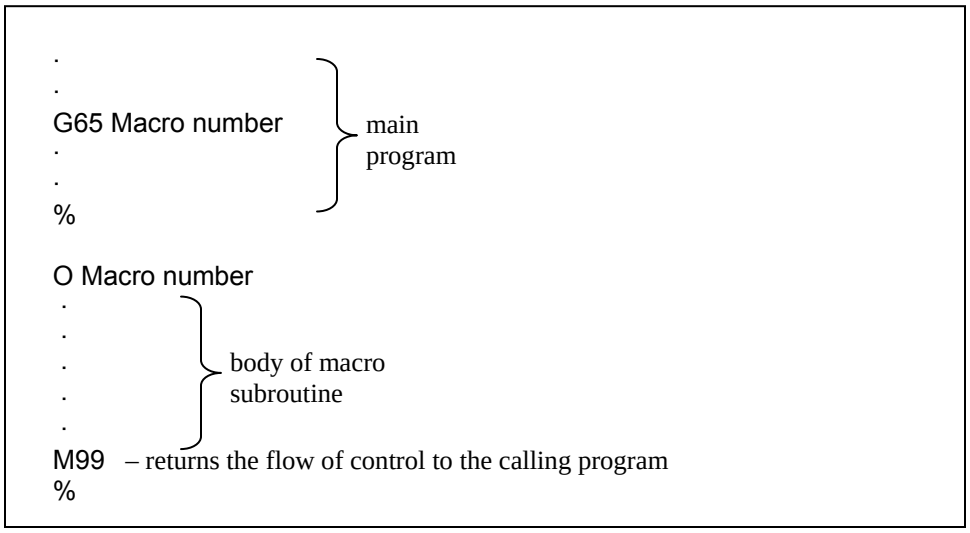


Figure 1-1: Format for a Macro Subroutine

NOTES:

1) Using M99 in the main part program returns the flow of control to the beginning of that main part program, resulting in an infinite loop.

2) Either the "%" symbol or a carriage return is required at the end of all NC and macro files.

1.3 Differences Between Macro Statements and NC Statements

Macro statements are blocks of code in a subprogram or a part program that must meet one of the following criteria:

1) Contains an arithmetic or logical operation (see *Chapter 4: Mathematical Operations and Logical Operations*)

2) Contains a control statement (see *Chapter 5: Flow of Control – Branching and Repetition*)

3) Contains a macro call command (see *Chapter 6: Calling Macro Programs from Within Part Programs*)

Any statement (block of code) not meeting the above criteria is considered to be an NC statement.

When a macro statement is executed, single block mode does not apply; i.e., the machine does not stop execution.

1.4 General Format Requirements

1) A space is required between keywords and values.

   For instance:

   GOTO99 → INVALID
   GOTO 99 → VALID

   [#11GT0] → INVALID
   [#11 GT 0] → VALID

2) Brackets "[ ]"must be used around conditional expressions and in mathematical and logical operations. Parentheses "( )"are invalid in these cases, and are used only for comments in the code.

   For instance:

   (#11 GT 0) → INVALID
   [#11 GT 0] → VALID

Chapter 2: Variables

2.1 Overview of Variables

The ServoWorks part programming language consists of blocks of code made up of F codes, G codes, M codes, N codes, S codes, T codes, etc. Both part programs and subprograms specify positions and distances associated with these codes directly. That is to say, the numerals that accompany the parameters for these F/G/M/N/S/T codes are exact physical dimensions, or exact feedrates, or scaling factors, etc.

The ServoWorks macro programming language allows you to specify either a numeric value or a variable number. When a variable number is used, the variable value can be assigned when the macro program is called from the part program, or the variable value can be calculated within the macro program, based on other numeric values or variables.

2.2 Types of Variables

There are six types of variables in the ServoWorks macro programming language:

1) The Null Variable (#0). The value of this variable is always null. You cannot assign a value to this variable.

> **⚠ CAUTION**
>
> DO NOT USE A NULL VARIABLE.
>
> Using a null variable (#0) in any line of code (in mathematical formulas, movement commands or conditional expressions) will cause an error, or unintended results.

2) Local Variables (#1-#99). Local variables are used within a local program scope, which may be a macro program, a subprogram or the main program. Local values are only held as long as the local scope exists. For example, both the main program and a macro program called from the main program can have a local variable #1, but they are separate variables with separate values that exist only within their scope. The value of local variables within a macro program is lost (initialized to null) when the macro program returns to the part program.

   You can assign values to local variables when you call a macro program from within a part program, or you can use local variables to hold the results of logical or mathematical operations.

3) Numbered Global Variables (#100-#499). Numbered global variables are used by the ServoWorks CNC system and are shared across multiple macro programs. The value is saved in the ServoWorks CNC system after a macro program (which may have assigned or changed the value of the global variable) returns to the part program. These variables lose their values when the control is completely shut down (powered off).

   Numbered global variables have file scope – that is, if a global variable is defined in a file (whether it is the main program, a subprogram, or a macro program; whether or not it is defined inside a subroutine), its value is only visible in that file.

4) Symbolic Global Variables (#axxx - #zxxx) . The ServoWorks CNC system allows the use of symbolic variables, with meaningful variable naming (such as #position), for the convenience of the user. These symbolic variables must be all lower case, preceded by a "#" sign, and contain no numbers.

   Symbolic global variables, like numbered global variables, are used by the ServoWorks CNC system and are shared across multiple macro programs. The value is saved in the ServoWorks CNC system after a macro program (which may have assigned or changed the value of the global variable) returns to the part program. These variables lose their values when the control is completely shut down (powered off). Symbolic global variables also have file scope.

   A symbolic global variable name can be up to 64 characters.

> **⚠ CAUTION**
>
> You will get unpredictable results if you specify a symbolic global variable that does not begin with "#," or that uses upper case letters or mixes upper and lower case letters, or that includes numbers in the name. Either the variable will be ignored or there will be an error, depending on whether or not the variable name conflicts with keywords.

5) Numbered Permanent Variables (#500-#999). Permanent variables are used by the ServoWorks CNC system and are shared across multiple macro programs. These variables keep their value in the ServoWorks CNC system after a macro program (which may have assigned or changed the value of the global variable) returns to the part program. They even keep their values even when the control is completely shut down (powered off), because the values are saved in the Windows registry. [NOTE: ServoWorks S-100T is the exception; the values of all numbered permanent variables are lost when the control is completely shut down (powered off).]

6) System Variables (#3000 - #13999). System variables are used for holding data/values related to NC operations, such as positions, feedrates, tool compensation values, workpiece coordinate offsets, etc. For more information, see *Chapter 3: System Variables*.

2.3 Specifying and Referencing Variables

2.3.1 Specifying a Variable

All variables except symbolic variables must be assigned a number. Symbolic variables should have lower case names.

All variables except symbolic variables must be specified with a number sign ("#") followed by the variable number. The correct format is:

```
#n = value or mathematical expression
```

where "n" is the variable index number. For instance, to specify the variable #1 and give it the value of 500, the line of code would simply be:

```
#1 = 500
```

To specify the symbolic variable #position and give it the value of 23.0, the line of code would simply be:

```
#position = 23.0
```

⚠ CAUTION

Parentheses are invalid for specifying a variable number with an expression, and would result in an error or in unpredictable results.

2.3.2 Referencing a Variable

You can reference a local, global, permanent or system variable in one of two ways:

1) If you want to directly use the value of a variable, put the variable number after a word address, such as "X#4". The value of variable #4 is used for the value of the part programming parameter "X."

   For example:

   ```
   #1 = 20
   #2 = 40
   #position = 23.5
   G01 X#1 Y#2 Z#position  → this is interpreted as "G01 X20 Y40 Z23.5"
   ```

2) If you want to use the value of a variable in an expression, enclose the expression in brackets ("[]") and put the expression after a word address, such as "X[#4+#5]." The value of variables #4 and #5 are added together, and used for the value of the part programming parameter "X."

You can reverse the sign of a variable value by referencing it with a minus sign (–) before the "#," such as "X–#4."

As an example, the following line in a macro program demonstrates several ways of referencing local, global or system variables:

```
G17 G02 X–#4 Y[#2–#1] R#20 F[#6+#7] Q#8
```

An example of referencing a symbolic variable follows:

```
#position = 100
G00 X#position Z#position
```

These lines would have the same effect as "G00 X100 Z100".

⚠ CAUTION

You cannot reference any variable with subprogram name calls (a "P" word address) or block/sequence numbers (an "N" word address).

For instance, either of the following lines will produce an error:

```
N#1 M98 P100
N0020 M98 P#1
```

⚠ CAUTION

VARIABLES MUST BE DEFINED/INITIALIZED BEFORE USE.

Using a null variable (#0) or an uninitialized variable in any line of code (in mathematical formulas, movement commands or conditional expressions) will cause an error, or unintended results.

2.4 Variable Values

2.4.1 Allowable Values for Variables

Local and global variables are stored as double-precision floating-point values. A double-precision floating-point number is a 64-bit approximation of a real number. The number can be zero or can range from -1.7976931348623157E+308 to -2.2250738585072014E-307 ($-1.7976931348623157 \times 10^{308}$ to $-2.2250738585072014 \times 10^{-307}$), or from 2.2250738585072014E-307 to 1.7976931348623157E+308 ($2.2250738585072014 \times 10^{-307}$ to $1.7976931348623157 \times 10^{308}$).

Values in ranges other than these will trigger an alarm in the ServoWorks CNC system.

When you assign a value without a decimal point, the value of the variable is assigned as a double-precision floating-point value with the maximum precision possible, approximately 15 digits of precision.

2.4.2 Rounding of Referenced Variables

When a variable is referenced and its value assigned to an address, the value is automatically rounded according to the maximum accuracy of double-precision floating-point numbers. A *floating-point calculation* is an arithmetic calculation done with floating-point numbers and often involves some approximation or rounding because the result of an operation may not be exactly representable.

Floating-point values are limited to a finite precision. For the purposes of the machine tool and factory automation industries, this highly accurate representation of values is more than adequate, and rounding of referenced values is undetectable. For a more complete explanation, see *Section 4.4: Precision and Errors*.

2.5 Processing Null and Uninitialized Variables

Any variable that has not been initialized (to which no value has been assigned) is undefined, or equal to the null variable (#0). This applies to undefined local, undefined global, undefined permanent, undefined symbolic or undefined system variables.

Using the null variable (#0) in any line of code will result in an error.

2.5.1 Uninitialized Variables in a Mathematical Formula

If a variable has the value of null and is used in a mathematical formula, the entire line of code is ignored, causing unintended results.

CAUTION

Using a null variable (#0) or an uninitialized variable in a mathematical formula will cause unintended results, and should be avoided.

2.5.2 Uninitialized Variables in a Movement Command

If a variable has the value of null and is used in a movement command, both the variable and the word address being referenced will be ignored, probably causing unintended results. For example:

```
#1 = 10
#2 = #0
G01 X#1 Y#2
```

"G01 X10" is the movement command that will be executed. The "Y" address will be ignored, because the value assigned to that address is undefined.

CAUTION

Using a null variable (#0) or an uninitialized variable in a movement command will be ignored, causing unintended results, and should be avoided.

2.5.3 Uninitialized Variables in a Conditional Expression

If a variable has the value of null and is used in a conditional expression, it will cause a syntax error.

CAUTION

Using a null variable (#0) or an uninitialized variable in a conditional expression will cause an error, and should be avoided.

Chapter 3: System Variables

3.1 Overview of System Variables

System variables are used for holding data/values related to NC operations, such as positions, feedrates, tool compensation values, etc. You can read internal NC data using system variables, and you can write to some, but not all system variables. System variables are how macro programs communicate with the PLC. As such, they are powerful and necessary for automation and for general-purpose part program development.

VARIABLE TYPE	RANGE	NOTES
Input from PLC (by bit)	#1000 – #1015	Read only
Input from PLC (16 bit word)	#1032	
Output to PLC (by bit)	#1100 – #1115	Read/Write (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*)
Output to PLC (16 bit word)	#1132	
Output to PLC (32 bit dword)	#1133	
1 millisecond timer	#3001	Read only
G-code modal groups	#4000 – #4032	
B code	#4102	
F code	#4109	
H code	#4111	
M code	#4113	
Sequence number	#4114	
S code	#4119	
T code	#4120	
Block end point position	#5001 – #5008	
Current position (machine)	#5021 – #5028	
Current position (work)	#5041 – #5048	
External work compensation	#5201 – #5208	• ServoWorks S-100T only • Read only
Work coordinate 1	#5221 – #5228	Read/Write (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*)
Work coordinate 2	#5241 – #5248	
Work coordinate 3	#5261 – #5268	
Work coordinate 4	#5281 – #5288	
Work coordinate 5	#5301 – #5308	
Work coordinate 6	#5321 – #5328	
Tool length wear compensation	#10001 – #10999	• ServoWorks MC-Quad, S-100M, S-120M and S-140M only • Read/Write (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*)
Tool length geometry compensation	#11001 – #11999	
Tool radius wear compensation	#12001 – #12999	
Tool radius geometry compensation	#13001 – #13999	

Table 3-1: Summary of System Variables

3.2 System Variables for Interfacing with the PLC

System variables #1100 – 1115, #1132 and #1133 (for interfacing with the PLC) can be read and written to (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*).

System variables for interfacing with the PLC are broken down as follows:

SYSTEM VARIABLE NUMBER	CORRESPONDING PLC ADDRESS	INTERFACE WITH PLC
#1000	G54.0	16-bit signal from PLC to macro. The signal is read bit by bit.
#1001	G54.1	
#1002	G54.2	
#1003	G54.3	
#1004	G54.4	
#1005	G54.5	
#1006	G54.6	
#1007	G54.7	
#1008	G55.0	
#1009	G55.1	
#1010	G55.2	
#1011	G55.3	
#1012	G55.4	
#1013	G55.5	
#1014	G55.6	
#1015	G55.7	
#1032	The 16-bit binary value of byte G54 and G55, G54 being the least significant byte.	16-bit signal from PLC to macro. All 16 bits of the signal are read at one time.

Table 3-2: System Variables for PLC Interface (1 of 2)

SYSTEM VARIABLE NUMBER	CORRESPONDING PLC ADDRESS	INTERFACE WITH PLC
#1100	F54.0	16-bit signal from macro to PLC. The signal is read bit by bit.
#1101	F54.1	
#1102	F54.2	
#1103	F54.3	
#1104	F54.4	
#1105	F54.5	
#1106	F54.6	
#1107	F54.7	
#1108	F55.0	
#1109	F55.1	
#1110	F55.2	
#1111	F55.3	
#1112	F55.4	
#1113	F55.5	
#1114	F55.6	
#1115	F55.7	
#1132	The 16-bit binary value of byte F54 and F55, F54 being the least significant byte.	16-bit signal from macro to PLC. All 16 bits of the signal are read at one time.
#1133	The 32-bit binary value of byte F56, F57, F58 and F59, F56 being the least significant byte.	32-bit signal from macro to PLC. All 32 bits of the signal are read at one time. Acceptable values range from –2,147,483,648 to +2,147,483,648.

Table 3-3: System Variables for PLC Interface (2 of 2)

3.3 System Variables for Tool Compensation

System variables for tool compensation values can be read and written to (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*).

System variable numbers for tool compensation values are available for ServoWorks MC-Quad, ServoWorks S-100M, ServoWorks S-120M and ServoWorks S-140M, and are broken down as follows:

TOOL NUMBER	TOOL LENGTH COMPENSATION		TOOL RADIUS COMPENSATION	
	GEOMETRY	WEAR	GEOMETRY	WEAR
1	#11001	#10001	#13001	#12001
. . .	. . .	. . .	. . .	. . .
200	#11200	#10200	#13200	#12200
. . .	. . .	. . .	. . .	. . .
999	#11999	#10999	#13999	#12999

Table 3-4: System Variable Numbers for Tool Compensations

3.4 System Variables for Timers

System variables for timers are broken down as follows:

SYSTEM VARIABLE NUMBER	TIMER DESCRIPTION
#3001	• Counts in 1-millisecond increments at all times • Resets to 0 each time the power is turned on • Resets to 0 each time 2,147,483,648 milliseconds is reached

Table 3-5: System Variables for Timers

3.5 System Variables for Modal Information

System variables for modal information specified in blocks immediately preceding the current block can be read. System variables for modal information can only be read, not written to.

⚠ CAUTION

If you try to write to a system variable for modal information, you will get an error. This variable is write-protected.

System variables for modal information for ServoWorks S-100T, ServoWorks MC-Quad, ServoWorks S-100M, ServoWorks S-120M and ServoWorks S-140M are explained as follows:

SYSTEM VARIABLE NUMBER	FUNCTION	MODAL GROUP
#4000	One shot G codes: G04, G09, G27, G28, G29, G30, G50, G52, G70~G76, G107	00
#4001	Movement commands: G00, G01, G02, G03, G32, G90, G92, G94	01
#4002	Constant surface speed control: G96, G97	02
#4005	Feed per minute / rotation per minute: G98, G99	05
#4006	Inch / metric: G20, G21	06
#4007	Tool radius compensation: G40, G41, G42	07
#4008	Spindle speed detect off: G25, G26	08
#4009	Stored stroke check: G22, G23	09
#4010	Canned cycle: G80, G83, G84, G85, G87, G88, G89	10
#4014	Workpiece coordinates: G53, G54~G59	14
#4015	Exact stop check mode and cutting mode: G61, G64	15
#4021	Polar coordinates compensation: G112, G113	21
#4102	B code	n/a
#4109	F code	n/a
#4113	M code	n/a
#4114	Sequence number	n/a
#4119	S code	n/a
#4120	T code	n/a

Table 3-6: System Variables for ServoWorks S-100T Modal Information

SYSTEM VARIABLE NUMBER	FUNCTION	MODAL GROUP
#4001	One shot G codes: G04, G05, G08, G09, G10, G27, G28, G29, G30, G31, G53, G65, G92	00
#4002	Movement commands: G00, G00.1, G01, G02, G03, G02.3, G03.3	01
#4003	Plane specification: G17, G18, G19	02
#4004	Absolute / incremental: G90, G91	03
#4006	Feed per minute / rotation per minute: G98, G99	05
#4007	Inch / millimeter: G20, G21	06
#4008	Tool radius compensation: G40, G41, G42	07
#4009	Tool length compensation: G43, G44, G49	08
#4010	Fixed cycle: G73, G74, G76, G80 – G89	09
#4011	I point / R point return: G98, G99	10
#4012	Scaling: G50, G51	11
#4015	Workpiece coordinates: G54-G59, G54.1	14
#4016	Cutting mode: G61, G64, G64.1	15
#4017	Coordinate system rotation: G68, G69	16
#4023	G50.1, G51.1	22
#4032	Linear interpolation feedrate include/exclude rotary axes: G310, G311	31
#4109	F code	n/a
#4111	H code	n/a
#4113	M code	n/a
#4114	Sequence number	n/a
#4119	S code	n/a
#4120	T code	n/a

Table 3-7: System Variables for ServoWorks MC-Quad, S-100M, S-120M, and S-140M Modal Information

As an example, the line of code "#5=#4006," the value of variable #5 would be either 20 or 21, depending upon whether G20 or G21 was active.

⚠ CAUTION

If you specify a system variable to read modal information that corresponds to a G code modal group that cannot be used (i.e. a G code related to ServoWorks S-100T functions when you're using ServoWorks MC-Quad), that system variable will be assigned a value of zero.

3.6 System Variables for Position Information

System variables for current positions can be read. System variables for position information can only be read, not written to.

⚠ CAUTION

If you try to write to a system variable for position information, you will get an error. This variable is write-protected.

System variables for position information are explained as follows:

SYSTEM VARIABLE NUMBER	POSITION INFORMATION	NOTES
#5001 – #5008 (for axes 1-8)	Block end point (the program position at the completion of the current block) in the current workpiece coordinate system[1]	• Tool compensation value not included • System variable can be read during movement
#5021 – #5028 (for axes 1-8)	Machine position (the command position measured from the machine origin) in the machine coordinate system	• Tool compensation value included • System variable can be read during movement, but care should be taken in how this data is used (due to the buffering / preread function[2])
#5041 – #5048 (for axes 1-8)	Program position (the desired position, specified by the program). This position is with respect to the workpiece coordinate system[1]	• Tool compensation value included • System variable can be read during movement, but care should be taken in how this data is used (due to the buffering / preread function[2])

Table 3-8: System Variables for Position Information

NOTES:

1) If no G54, G55, G56, G57, G58 or G59 code has been programmed, the workpiece coordinate system will be the same as the machine coordinate system.

2) See *Section 7.1: Preread Function / Block Buffering – Problems and Workaround*.

An example follows, where X is Axis #1, Y is Axis #2, and the G54 workpiece coordinates are X=100 and Y=100.

```
G54  (sets the G54 workpiece coordinate system)
G01 X10 Y10  (moves the X and Y coordinates to 10.000, 10.000 in the G54
workpiece coordinate system)
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10
#1=#5001
#2=#5021
#3=#5041
```

Dummy delay blocks to ensure proper evaluation of the local variables (#1, #2 and #3) used in the following lines of code – see *Section 7.1: Preread Function/Block Buffering – Problems and Workaround*

At the completion of the execution of the above code, the value of #1 would be 10.000 (the program position relative to the G54 workpiece coordinate system). The value of #2 would be 110.000 (the machine position relative to the machine coordinate system). The value of #3 would be 10.000 mm (the program position relative to the G54 workpiece coordinate system).

3.7 System Variables for Workpiece Coordinates

System variables for workpiece coordinates can be read and written to (with some limitations – see *Section 7.1: Preread Function/ Block Buffering – Problem and Workaround*).

System variables for workpiece coordinates in various workpiece coordinate systems are explained as follows:

SYSTEM VARIABLE NUMBER	EXPLANATION
#5201 – #5208 (for axes 1-8)	External workpiece coordinates
#5221 – #5228 (for axes 1-8)	G54 workpiece coordinates for workpiece coordinate system #1
#5241 – #5248 (for axes 1-8)	G55 workpiece coordinates for workpiece coordinate system #2
#5261 – #5268 (for axes 1-8)	G56 workpiece coordinates for workpiece coordinate system #3
#5281 – #5288 (for axes 1-8)	G57 workpiece coordinates for workpiece coordinate system #4
#5301 – #5308 (for axes 1-8)	G58 workpiece coordinates for workpiece coordinate system #5
#5321 – #5328 (for axes 1-8)	G59 workpiece coordinates for workpiece coordinate system #6

Table 3-9: System Variables for Workpiece Coordinates

Chapter 4: Mathematical Operations and Logical Operations

4.1 Summary of Mathematical and Logical Operations

The ServoWorks macro programming language includes many mathematical operations and logical operations to facilitate your macro programming.

Tables 4-1, 4-2 and 4-3 summarize the mathematical and logical operations can be performed on variables. In the tables, the expression to the right of the operator ("=") can contain constants, variables, or an expression containing constants and/or variables. Variables to the left of the operator ("=") can also be replaced with an expression. #A, #B and #C represent local, global, permanent, system or symbolic variables, unless noted otherwise.

FUNCTION	FORMAT	NOTES
Assignment	#A = #B	
Sum	#A = #B + #C	
Difference	#A = #B - #C	
Product	#A = #B * #C	
Quotient	#A = #B / #C	
Exponent	#A = #B ^ #C	
Sine	#A = sin [#B]	• #B and #C should be specified in units of degrees. 45 degrees and 30 minutes should be represented as 45.5°. • Function "tan" performs "sin/cos".
Arcsine	#A = asin [#B / #C]	
Cosine	#A = cos [#B]	
Arccosine	#A = acos [#B / #C]	
Tangent	#A = tan [#B]	
Arctangent	#A = atan [#B / #C]	

Table 4-1: Summary of Mathematical and Logical Operations (1 of 3)

⚠ CAUTION

If you input data in radians instead of degrees for trigonometric functions you will get an error.

FUNCTION	FORMAT	NOTES
Square Root	#A = sqrt [#B]	
Absolute Value	#A = abs [#B]	
Rounding Off Below the Decimal Point	#A = round [#B]	• When the "round" function is included in an arithmetic or logic operation command, in an IF statement, or in a WHILE statement, the "round" function rounds off at the first decimal place. (If #10 = 6.6666, then #1 = round [#10] changes the value of variable #10 to 6.0.) • When the "round" function is used in an NC statement address, the "round" function rounds off the specified value according to the least input increment of the address. (If the least input increment is 0.001 mm, and #1 = 5.6687, then G00 X#1 moves axis X 5.669 mm.)
Rounding Down (Absolute Value)	#A = fix [#B]	• Integers are rounded up or down according to their absolute values, not their actual values. Therefore, you must be very careful when rounding negative numbers up or down. • Rounding up an integer: the absolute value of the integer produced by the "fup" function is greater than the absolute value of the original number. • Rounding down an integer: the absolute value of the integer produced by the "fix" function is less than the absolute value of the original number. • Examples: Say that #10 = 3.1 and #20 = –3.1. #30 = fup [#10] ⇒ #30 = 4.0 #30 = fix [#10] ⇒ #30 = 3.0 #30 = fup [#20] ⇒ #30 = –4.0 #30 = fix [#20] ⇒ #30 = –3.0
Rounding Up (Absolute Value)	#A = fup [#B]	

Table 4-2: Summary of Mathematical and Logical Operations (2 of 3)

FUNCTION	FORMAT	NOTES
Natural Logarithm	#A = ln [#B]	
Exponential Function	#A = exp [#B]	• Relative error may become 10^{-8} or greater • When #A (the result of the operation) exceeds 3.65 x 10^{47} (for #B ≈ 110), an overflow occurs, you will get an error

Or	#A = #B or #C	• These logical operations are performed on binary numbers bit by bit
		• "not" is implemented as follows:
		  • For a conditional statement, "not" changes the value from "TRUE" to "FALSE," or from "FALSE" to "TRUE"
		  • For the number "0," "0" is evaluated as "FALSE," and the "not" operation then returns the value of "TRUE"
		  • For any non-zero numeric value (decimal value or binary), the numeric value is evaluated as "TRUE," and the "not" operation then returns the value of "FALSE"
		  • Examples:
		  0 → FALSE
		  !0 → TRUE
		  6 → TRUE
		  !6 → FALSE

Xor	#A = #B xor #C	
And	#A = #B and #C	
Not	#A = not #B, #A = !#B	

Table 4-3: Summary of Mathematical and Logical Operations (3 of 3)

4.2 Format of Mathematical and Logical Operations

There are three important rules for the formatting of mathematical and logical operations:

1) All mathematical functions must be specified in lower case: sin [#B].

2) Brackets must be used: sin **[**#B**]**. The use of parenthesis (i.e. "sin (#B)") would result in an error.

3) To include comments in mathematical and logical operations, use parenthesis ("( )").

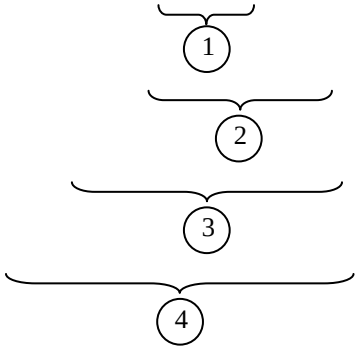
4.3 Order of Operations

4.3.1 Nesting

In addition to using brackets as part of functions, you can use brackets to specify a certain order of operations. There is no limit to the number of folds for nested brackets. For equations with brackets, expressions and functions within brackets are executed from the innermost to the outermost pairs of brackets.

For instance:

#1 = cos [#3 + [[#4 / #2] * #5]]



(1), (2), (3) and (4) indicate the order of operations.

4.3.2 Priority of Operations

Commands including mathematical and logical operations are executed in a predictable order, as follows:

1) Functions (from left to right)

2) Multiplication and division operations (*, /, and) (from left to right)

3) Addition and subtraction operations (+, –, or, xor) (from left to right)

4.4 Precision and Errors

4.4.1 Precision and Errors Related to Mathematical and Logical Operations

There is always some level of error in the execution of mathematical and logical operations. However, all ServoWorks CNC systems use highly precise double-precision floating-point variables, so the level of error is quite small, approximately 15 digits.

The 64-bit double-precision floating point standard is described by IEEE. A double-precision floating-point word has the following internal format:

The first bit is the sign bit, S, the next eleven bits are the exponent bits, 'E', and the final 52 bits are the fraction 'F'. A number *n* is expressed in floating point format as:

$$n = (-1)^s * m * 2^e$$

where “s” is the value of the sign bit, “m” is the mantissa, and “e” is the exponent.

The mantissa m is "normalized," which means that it is always scaled such that it is greater than or equal to 1, and less than 2. Therefore, the ones bit (2^0) is always set, and is not present in the actual number. This is called an *implied* bit. Since the mantissa is 52 bits, plus the implied ones bit, the precision of the number is stored to 53 bits, or 2^{53} = 900,719,925,474,100, approximately 15 digits of precision (2.2250738585072014E-307, or $2.2250738585072014 \times 10^{-307}$).

The effective range (excluding infinite values) of IEEE double-precision floating-point numbers is:

- Binary: $\pm (2\text{-}2^{-52})^{1023}$
- Decimal: $\sim \pm 10^{308.25}$ (This value is the end point of the range with the IEEE-754 round-to-nearest value mode applied.)

The maximum value of the floating-point number is about $1.8*(10^{308})$.

Floating-point numbers usually behave very similarly to the real numbers they are used to approximate. However, there are some cases where floating-point numbers do not model real numbers well because most floating-point values can't be precisely represented as a finite binary value. For example 0.1 is .0001100110011... in binary (it repeats forever), so it can't be represented with complete accuracy on a computer using binary arithmetic, which includes all PCs. Similarly, the result of arithmetic operations may need to be rounded. For example, 2/3 might yield 0.666666666666667.

Because of this, you can’t assume that a result is accurate to the last decimal place. There are always small differences between the "true" answer and what can be calculated with the finite precision of any floating point processing unit. Cumulative rounding errors are possible, but are unlikely to ever be problematic.

⚠ CAUTION

Any time you specify division by zero, you will get an error.

4.4.2 Precision and Errors Related to Conditional Expressions

Conditional expressions using EQ (equal to), NE (not equal to), GE (greater than or equal to), GT (greater than), LE (less than or equal to) or LT (less than), can also result in errors.

As an example, if you are using the expression "IF [#10 EQ #20]," if there are small errors in how variables #10 and #20 have been calculated, you may get a FALSE evaluation when a TRUE evaluation would have been expected. Also, consider that both variables #10 and #20 may have been rounded off slightly, which could also affect the evaluation.

To avoid this problem, you could replace the expression "IF [#10 EQ #20]" with "IF [abs [#10 – #20] LT 0.00001," thus assuming that two values are considered to be equal if their difference does not exceed some allowable limit (0.00001 in this case)

4.4.3 Errors Related to Null Variables or Uninitialized Variables

Any time you use a null variable or an uninitialized variable, you will get an error or unintended results.
.

Chapter 5: Flow of Control – Branching and Repetition

5.1 Overview of Flow Control

There are five types of branch and repetition statements that can be used to direct the flow of control in a macro program:

1) The GOTO statement (an unconditional branch statement).

2) The IF GOTO statement (a conditional branch statement).

3) The IF THEN statement (a conditional execution statement).

4) The IF ELSE ENDIF statement (a conditional execution with branching statement).

5) The WHILE DO END statement (conditional loop: repetition while certain conditions are met).

These branching and repetition statements are case insensitive: they can be both lower case or upper case ("IF ELSE ENDIF" or "if else endif."

5.2 GOTO Statement (Unconditional Branching)

The GOTO statement is a very simple statement that directs the execution to a specific sequence number in the range of 1 to 99,999.

The format of a GOTO statement is:

GOTO n ← NOTE: a space is required between "GOTO" and "n," the sequence number

"n" refers to the sequence number. It can be specified in one of two ways:

1) As a direct number. (GOTO 1000)

2) In the format "NXXXX" (GOTO N1000)

⚠ CAUTION

It is strongly recommended that you use GOTO statements to branch to sequence numbers that occur later in the macro program, rather than a sequence number that comes BEFORE the GOTO statement. In other words, the flow of control should always be in the forward direction, not in the reverse direction, or it will impact processing time.

⚠ CAUTION

If you specify a sequence number outside the range of 1 to 99,999, you will get an error.

⚠ CAUTION

It is invalid to use a variable or an expression as a sequence number. The following lines of code would each result in an error:

```
GOTO #1
GOTO [#1 + #2]
```

5.3 Conditional Statements and Comparison Operators

A conditional statement is an expression that compares two values using comparison operators. These values may be constants, variables, or expressions. Conditional statements are evaluated to be either TRUE or FALSE.

A non-zero numeric value is evaluated as TRUE. A zero value is evaluated as FALSE.
An example follows:

```
#1 = 100
#2 = 0
IF [#1]  ← evaluates as TRUE
IF [#2]  ← evaluates as FALSE
```

Comparison is done using operators summarized in the following table:

COMPARISON OPERATOR		EXPLANATION	ACCEPTABLE FORMATS
EQ	**=**	Equal to	[#A EQ #B] [#A = #B]
NE	**≠**	Not equal to	[#A NE #B]
GT	**>**	Greater than	[#A GT #B] [#A > #B]
GE	**≥**	Greater than or equal to	[#A GE #B] [#A >= #B]
LT	**<**	Less than	[#A LT #B] [#A < #B]
LE	**≤**	Less than or equal to	[#A LE #B] [#A <= #B]

Table 5-1: Summary of Comparison Operators

⚠ CAUTION

Conditional statements that include the null variable (#0) will result in a syntax error.

⚠ CAUTION

Brackets must be used: [#A EQ #B**]**. The use of parenthesis (i.e. "(#A EQ #B)") would result in an error.

5.4 IF Statement

The IF statement specifies some conditional expression. It may be followed by a GOTO statement, a THEN statement, or an ELSE…ENDIF structure.

5.4.1 IF GOTO (Conditional Branching)

The format of an IF GOTO statement is:

IF [<conditional expression>] GOTO n where "n" is the sequence number.

If that conditional expression is satisfied, program execution branches to sequence number n. If that conditional expression is not satisfied, program execution continues with the execution of the statement following the IF GOTO statement.

As noted previously, "n" refers to the sequence number, and can be specified in one of two ways:

1) As a direct number. (GOTO 1000)

2) In the format "NXXXX" (GOTO N1000)

⚠ CAUTION

If you specify a sequence number outside the range of 1 to 99,999, you will get an error.

⚠ CAUTION

It is invalid to use a variable or an expression as a sequence number. The following lines of code would each result in an error:

```
GOTO #1
GOTO [#1 + #2]
```

5.4.2 IF THEN (Conditional Execution)

The format of an IF THEN statement is:

IF [<conditional expression>] THEN <macro statement>

If that conditional expression is satisfied, the program executes the macro statement following the "THEN." If that conditional expression is not satisfied, program execution continues with the execution of the statement following the IF THEN statement (and the macro statement following the "THEN" is not executed).

⚠ CAUTION

The macro statement needs to be in the same line as the IF THEN statement, or you will get an error.

5.4.3 IF ELSE ENDIF (Conditional Execution with Branching)

The format of an IF ELSE ENDIF structure is:

```
IF [<conditional expression>]
<macro statement(s)>
ELSE
<macro statement(s)>
ENDIF
```

If the first conditional expression is satisfied, the program executes the macro statement(s) following "IF" (and the macro statement(s) following the "ELSE" are not executed). If that first conditional expression is not satisfied, then program execution continues with the execution of the second macro statement or set of macro statements following "ELSE" (and the macro statement(s) following the "IF" are not executed). Program execution then continues with the execution of the statement following the ENDIF statement.

Only one of the two sets of macro statements in an IF ELSE ENDIF structure is executed.

5.5 WHILE DO END Statement (Conditional Looping)

The WHILE DO END statement is used to specify repetition for as long as some conditional expression is met. When the conditional expression is satisfied, the blocks of code from the "DO" to the "END" statements are executed. If the conditional expression is not satisfied, the blocks of code from the "DO" to the "END" statements are not executed, and program execution continues with the block of code immediately following the "END" statement.

The format of a WHILE DO END statement is:

```
WHILE [<conditional expression>] DO m
.
.
.
END m          where m is an identification number for specifying the range of execution.
```

"m" numbers can be reused as many times as required:

```
WHILE [<conditional expression>] DO 1
.
.
.
END 1

WHILE [<conditional expression>] DO 1
.
.
.
END 1
```

⚠ CAUTION

If you specify an "m" number outside the range of 1 to 99,999, you will get an error.

WHILE/DO/END statements can be consecutive, or can be nested inside of each other. See *Section 5.6: Nesting* for examples.

5.6 Nesting

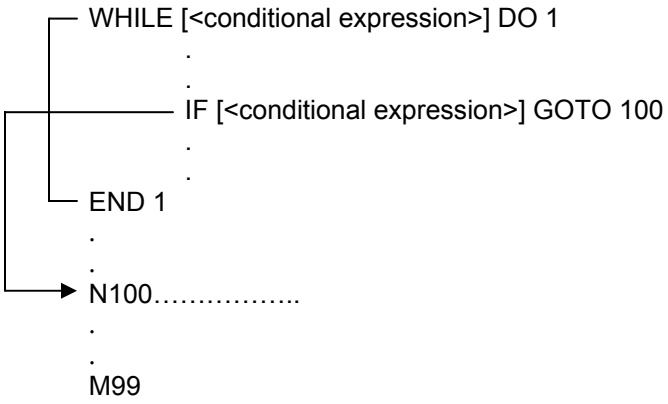
5.6.1 Acceptable Nesting

Conditional statements and repetition statements can be nested within each other, with certain logical restrictions. IF ELSE ENDIF and WHILE DO END statements can be nested with no limit to the depth of nesting.

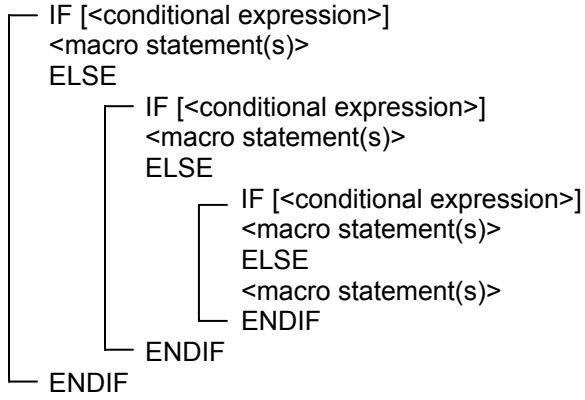
An example of acceptable nesting of WHILE DO END statements follows:

```
WHILE [<conditional expression>] DO 1
    .
    .
    WHILE [<conditional expression>] DO 2
        .
        .
        WHILE [<conditional expression>] DO 3
            .
            .
        END 3
        .
        .
    END 2
    .
    .
END 1
```

The flow of control can be directed to outside of the WHILE DO END loop, an example of which follows:



An example of acceptable nesting of an IF ELSE ENDIF statement follows:



5.6.2 Unacceptable Nesting

There are several nesting conditions that that would cause an error in the execution of your macro program:

1) Overlapping WHILE DO END ranges of WHILE statements

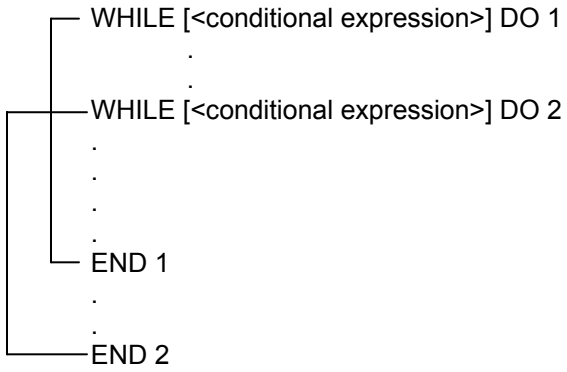
2) GOTO statements that branch the flow of control within a DO/END Loop of a WHILE statement

3) Infinite loops

5.6.2.1 Overlapping DO Ranges of WHILE Statements

It is imperative that DO/END ranges do not overlap. Overlapping DO ranges cannot logically be executed.

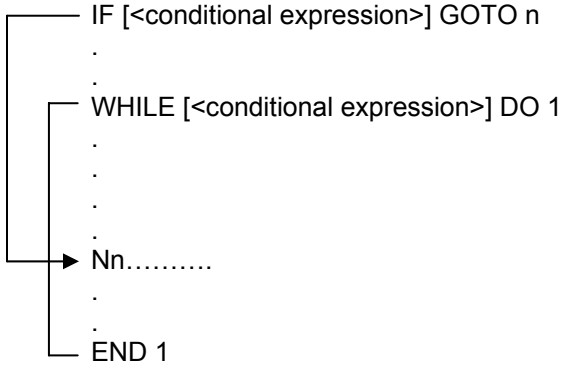
An example of overlapping DO ranges follows:



5.6.2.2 GOTO Statements That Branch The Flow Of Control Within A DO END Loop Of A WHILE Statement

DO/END loops cannot be entered except from the while statement. If you attempt to use a GOTO statement to send the flow of control to within a DO/END loop, you will get an error, as this cannot logically be executed (the evaluation of the conditional statement of the WHILE loop is skipped).

An example of a GOTO statement that branches the flow of control within a DO/END loop follows:



5.6.2.3 Infinite Loops

Infinite loops will produce errors and unexpected program execution. Some conditions under which infinite loops are created include the following:

- The conditional expression for a WHILE statement is always true – none of the variables within the conditional expression are modified by statements within the DO END loop
- A DO END loop which doesn't follow a WHILE statement
- Including M99 in the main program

Chapter 6: Calling Macro Programs

6.1 Overview of Macro Calls

There are differences between calling macro subprograms and calling macro subroutines from within part programs.

Macro subprograms can be called using custom G, M, S or T codes, or by using the M98 subprogram call, in the format:

M98 P R where P is the subprogram name, and R specifies the number of times the subprogram should be executed.

Calls to macro subprograms using M98 have the disadvantage of not allowing the passing of variables.

For example, "M98 PTest_1 R3" calls subprogram OTest_1.dat in the same folder as the parent program, and repeats it three times. The subprogram must be in a separate file from the part program file in which the M98 subprogram call appears. The subprogram file must be in the same folder as the part program file.

Macro programs can be called from within part programs with any of the following methods:

1) A subprogram macro call (M98) } Macro programs must be in the same folder as the part program file.
2) A simple macro call (G65) } Macro subroutines must be in the same file as the main program
3) Custom G code macro call
4) Custom M code macro call
5) Custom S code macro call
6) Custom T code macro call

} Macro programs may be anywhere, even on the network

The differences between an M98 subprogram call and other macro calls are summarized in the following table:

FEATURE	M98 SUBPROGRAM CALL	G65 SIMPLE MACRO CALL	CUSTOM G AND M MACRO CALLS	CUSTOM S AND T MACRO CALLS
Arguments can be specified (data passed)	NO	YES	YES	YES
Calls the macro…	Calls the macro subprogram from another file in the same folder where the main program resides	Calls the macro subroutine from within the same file as the main program	Calls the macro subprogram from another file in the specified macro program folder (specified with macro parameters)	Calls the macro subprogram from another file in the specified macro program folder (specified with macro parameters)

Table 6-1: Summary of Differences Between M98 Subprogram Call and Macro Calls

6.2 Argument Assignment

When passing arguments to macro programs, we strongly recommend using decimal points. When arguments are passed without decimal points, the units used are assumed to correspond to the least input increment of each address, which may vary according to the configuration of the ServoWorks CNC system.

For example, the statement "G65 P9000 A10. B20. C30." assigns the value of "10" to #1, the value of "20" to #2 and the value of "30" to #3. In other words, addresses A, B and C are used to pass arguments, which are assigned to local variables #1, #2 and #3.

The argument assignment protocol for ServoWorks CNC macros uses all of the letters of the alphabet (once each) except G, L, N, O and P.

⚠ CAUTION

If you specify a G, L, N, O or P in your argument assignments, you will get an error.

Addresses that need not be specified (that won't be used by the macro program) can be omitted from the argument assignments. The values of local variables corresponding to omitted addresses are set to null.

The letters and the local variables to which the letters are assigned are summarized in the following table:

ADDRESS LETTER	VARIABLE NUMBER
A	#1
B	#2
C	#3
D	#7
E	#8
F	#9
H	#11
I	#4
J	#5
K	#6
M	#13
Q	#17
R	#18
S	#19
T	#20
U	#21
V	#22
W	#23
X	#24
Y	#25
Z	#26

Table 6-2: Summary of Argument Assignment Protocol for ServoWorks CNC Macros

An example follows: the statement "G65 P9000 A10. B20. F2. S100." assigns the value of "10" to #1, the value of "20" to #2, the value of "2" to #9 and the value of "100" to #19. In other words, addresses A, B, F and S are used to pass arguments, which are assigned to local variables #1, #2, #9 and #19. The values of local variables #3, #4, #5, #6, #7, #8, #11, #13, #17, #18, #20, #21, #22, #23, #24, #25 and #26 are set to null because the addresses that correspond with them were omitted from the argument assignment statement.

6.3 Simple Macro Call (G65)

A simple macro call is a one-shot command in which a macro is called once using G65 (although it may be executed more than once, depending upon the L parameter, the number of times the macro is to be repeated).

Required Format

G65 P L <argument assignment>

Possible Parameters That Can Be Used With G65

P – macro number (subroutine number)
L – number of times to repeat the execution of the macro (from 1 to 9,999 – 1 by default)

In addition to these two parameters, the macros are called with argument assignments, which are not considered to be parameters.

Example 1

G65 P100 L3 Z20.0 R2.5 F500 – calls macro program O100.dat, which will be repeated 3 times, and assigns the value 20.0 to address Z, 2.5 to address R and 500 to address F

O100.dat – this line indicates the macro program O100.dat, which is in the same file, and which precedes the subroutine code

Example 2

```
G28
G50 X0 Z0
G65 P1 Z-5 R.5 Q1 D0.3 F20.0
M30
%

O1
IF [#26-#18>0] GOTO 100
IF [#17<=0] GOTO 100
IF [#7<0] GOTO 100
IF [#9<=0] GOTO 100
#50 = #18
G00 X0
G00 Z#50
WHILE [#50-#17 > #26] DO 1
#50 = #50 - #17
G01 Z#50 F#9
#50 = #50 + #7
G00 Z#50
#50 = #50 - #7
END 1
G01 Z#26 F#9
G00 Z#18
N100
M99
%
```

main program

macro subroutine

Limitations

- G65 must be specified before any argument.
- The macro program (the subroutine) must be in the same file as the part program file that calls the macro program.

6.4 G65 Macro Call Nesting

Macro calls can be nested to a depth of four levels. (This does not include M98 subprogram calls.)

There are local variable levels (0 to 4) pertaining to nesting. The level of the main program is level 0. Each time a macro is called, the level of the local variable is incremented by one, and the values of the local variables at the previous level are saved in the ServoWorks CNC system.

When M99 is executed in a macro program (indicating the end of the macro program), control returns to the calling program, the local variable level is decremented by one, and the values of the local variables that existed when the macro was called are restored.

⚠ CAUTION

When M99 is executed in a main program, it results in an infinite loop.

6.5 Custom Macro Calls Using G Codes, M Codes, S Codes or T Codes

6.5.1 Overview

You can create customized G, M, S or T codes to call macro programs, using Configuration Mode of one of the ServoWorks S-100M series (S-100M, S-120M and S-140M). This is very similar to the G65 simple macro call. Essentially, you can use this capability to create your own special, user-defined G, M, S or T code functions (written as customized macro programs) and call them using your own G, M, S or T code number.

You can define G, M, S or T codes as specialized codes by using macro parameters to associate these codes with specific macro programs. For S and T codes, the specified macro program applies to ALL S and T codes. For G and M codes, you define specific G and M codes (up to ten of each) that you associate with different macro programs.

When the "Enable Custom G/M/S/T Macro Calls" parameter (set in Configuration Mode of ServoWorks S-100M, S-120M or S-140M) is set to "Enabled," the ServoWorks RealTime DLL searches part program files before starting G code execution and generates (creates) a temporary file (with a user-defined file name and location), in which every special G, M, S and T code (defined by the Macro function parameters) is replaced with a G65 simple macro call to the specified macro program, and each macro program is appended to this temporary file as a subroutine. S and T codes are kept as parameters (arguments) in any new line involving special S or T codes that invoke a macro program.

NOTE: When the "Enable Custom G/M/S/T Macro Calls" parameter is set to "Enabled," the temporary file is created and executed even if there are no special G, M, S or T codes in the part program.

6.5.2 Macro Call Function Parameters for Custom G Codes, M Codes, S Codes or T Codes

6.5.2.1 Enable Custom G/M/S/T Macro Calls

Description

Enabling or disabling of custom G, M, S and T macro calls.

Valid Values: 0, 1

Meaning of Values

0 – Custom G, M, S and T code macro calls disabled
1 – Custom G, M, S and T code macro calls enabled

Default Value: 0

6.5.2.2 Macro Program Folder (Full Path)

Description

The directory (folder) where all the user-created macro programs related to custom G, M, S or T code macro calls must be stored. The ServoWorks CNC controller will search this folder for all macro programs.

Required Format

You must include the full path for this folder, including the drive.

Example

C:\Program Files\SoftServo\S-100M\ncdata\Macro\

Notes

The path for this folder can be on a local drive or on a network.

Limitations

This folder must already exist. If you type the path of a folder that does not exist, you will get an error message similar to the following:

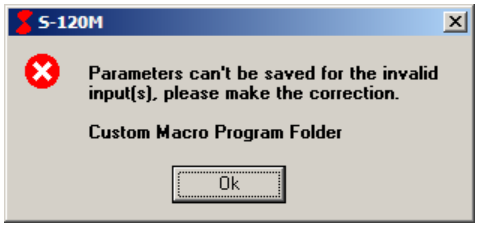


Figure 6-1: Error in Naming Macro Program Folder

6.5.2.3 Output File Name (Full Path)

Description

The name and location for the temporary NC program file that is created by and executed by the ServoWorks G-Code Parser. This temporary file consists of the part program file with the following changes:

- All lines of code that contain special G, M, S or T codes are replaced with G65 simple macro calls to the corresponding user-defined macro programs
- These user-defined macro programs are appended to this temporary part program file as macro subroutines

Required Format

You must include the full path for this file, including the drive.

Example

C:\Program Files\SoftServo\S-100M\ncdata\Macro\tempdata.dat

Limitations

- The location for this temporary file must be in a local folder, not on a network or a non-system drive.
- This folder must already exist. If you type the path of a folder that does not exist, you will get an error message similar to the following:

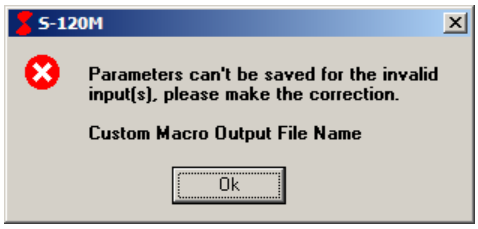


Figure 6-2: Error in Naming Macro Output File Name

Notes

- When the "Enable Custom G/M/S/T Macro Calls" is set to "Enabled," this temporary file is created whether or not there are any special G, M, S or T codes.
- Because this temporary part program file is executed by the ServoWorks G-Code parser and not the original part program file, M98 blocks will look for subprograms in the folder containing the temporary file and not the original part program file. To have M98 blocks look for subprograms in the folder containing the original part program file, set the output file to be in the same folder as the original part program file.

6.5.2.4 S Code Setting

Description

The name of the macro program to be called for each instance of an S code in the user's part program.

Required Format

An integer between 1 and 999999999 (with no commas).

Example

3333 (This would cause all S codes to invoke a macro program file named "O3333.dat")

Limitations

- You cannot call a macro program with an S code from within a macro program. All S codes in macro programs are treated as ordinary S codes.

  For example, if S1000 is defined as a custom S code macro call to O1234.dat, if "S500" appears in O1234.dat, it will NOT be treated as a macro call.

- No nesting of calls using S codes is allowed. In other words, if S codes are defined to call a macro program, any S codes in that macro program (subroutine) will be treated as ordinary S codes, and will not call that macro program again.

- All S codes in subprograms called with M98 are treated as ordinary S codes.

Notes

The S code is kept as a parameter (argument) in the new line that invokes the macro program. For example, the line of code "S10" would be replaced with "G65 P3333 S10."

6.5.2.5 T Code Setting

Description

The name of the macro program to be called for each instance of a T code in the user's part program.

Required Format

An integer between 1 and 999999999 (with no commas).

Example

3333 (This would cause all T codes to invoke a macro program file named "O3333.dat")

Limitations

- You cannot call a macro program with a T code from within a macro program. All T codes in macro programs are treated as ordinary T codes.

  For example, if T1000 is defined as a custom T code macro call to O1234.dat, if "T500" appears in O1234.dat, it will NOT be treated as a macro call.

- No nesting of calls using T codes is allowed. In other words, if T codes are defined to call a macro program, any T codes in that macro program (subroutine) will be treated as ordinary T codes, and will not call that macro program again.

- All T codes in subprograms called with M98 are treated as ordinary T codes.

Notes

The T code is kept as a parameter (argument) in the new line that invokes the macro program. For example, the line of code "T10" in the original NC part program would be replaced with "G65 P3333 T10."

In the O3333.dat macro program, the code should be written using the variable "T#20," because "#20" is the variable number that corresponds to address letter T (see Table 6-2). When the parameter "T10" is passed into the O3333.dat macro program as an argument, T#20 will be replaced with T10. In this way, the O3333.dat macro program can support all T codes, with the T codes defined by the tool number called in the original NC program.

6.5.2.6 M Code Settings

Description

The M code settings actually have two parts:

1) The name of the M code that invokes a macro program file (that defines this custom M code).

2) The name of the macro program to be called for each instance of that specific M code in the user's part program.

Required Format

The format for an M code can be an integer, or an integer with one digit after the decimal point. This is helpful, since you can name special M codes as non-integer numbers to set them apart from normal, predefined M codes which are always an integer value.

The name of the macro program should be an integer between 1 and 999999999 (with no commas).

Example

The name of the M code could be "3" (which defines "M03" as a customized M code), or could be "3.1" (which defines "M03.1" or "M3.1" as a customized M code).

The name of the macro program could be "3333" (which would cause the special M code to invoke a macro program file named "O3333.dat")

Limitations

- No nesting of calls using any specially defined M codes is allowed. In other words, if a special M code is defined to call a macro program, any instances of that same M code in that macro program (subroutine) will be treated as an ordinary M code, and will not call that macro program again.

  However, if a DIFFERENT special M code is used in a macro program (different than the special M code that called that macro program), then it WILL call its associated macro program from within the macro program.

  For example, if M3.1 is defined as a custom M code macro call to O3131.dat, if "M3.1" appears in O3131.dat, it will NOT be treated as a macro call. But if M4.1 is defined as a custom M code macro call to O4141.dat, and M4.1 appears in O3131.dat, it WILL call O4141.dat from O3131.dat.

- All M codes in subprograms called with M98 are treated as ordinary M codes.

6.5.2.7 G Code Settings

Description

The G code settings actually have two parts:

1) The name of the G code that invokes a macro program file (that defines this custom G code).

2) The name of the macro program to be called for each instance of that specific G code in the user's part program.

Required Format

The format for a G code can be an integer, or an integer with one digit after the decimal point. This is helpful, since you can name special M codes as non-integer numbers to set them apart from normal, predefined M codes which are always an integer value.

The name of the macro program should be an integer between 1 and 999999999 (with no commas).

Example

The name of the G code could be "23" (which defines "G23" as a customized G code), or could be "23.1" (which defines "G23.1" as a customized G code).

The name of the macro program could be "3333" (which would cause the special G code to invoke a macro program file named "O3333.dat")

Limitations

- No nesting of calls using any specially defined G codes is allowed. In other words, if a special G code is defined to call a macro program, any instances of that same G code in that macro program (subroutine) will be treated as an ordinary G code, and will not call that macro program again.

  However, if a DIFFERENT special G code is used in a macro program (different than the special G code that called that macro program), then it WILL call its associated macro program from within the macro program.

  For example, if G23.1 is defined as a custom G code macro call to O231.dat, if "G23.1" appears in O231.dat, it will NOT be treated as a macro call. But if G24.1 is defined as a custom G code macro call to O241.dat, and G24.1 appears in O231.dat, it WILL call O241.dat from O231.dat.

- All G codes in subprograms called with M98 are treated as ordinary G codes.

Chapter 7: Macro Statement Processing

7.1 Preread Function/Block Buffering – Problems and Workaround

The ServoWorks CNC Engine communicates with the ServoWorks G-Code Parser, and determines the timing of the ServoWorks G-Code Parser. There is a buffer of five blocks of binary data (each the result of parsing one block of code from a part program) between the ServoWorks CNC Engine and the ServoWorks G-Code Parser. This buffer is necessary to provide the ServoWorks CNC Engine with multiple blocks each service routine. However, this "5 block preview" also causes some problems related to the updating of certain system variables.

There are occasions where macro statements using system variables are evaluated, parsed, and delivered to the block buffer BEFORE the previous lines have been executed, and the relevant system parameters have been updated. Therefore, the evaluation of a macro statement containing a system variable may not evaluate as expected.

For example, let's say the current position of the X axis is 0.0, and then the following five blocks of code are each parsed and sent to the block buffer as a group:

```
G01 X10
G01 X20
G01 X30
G01 X40
IF [#5021 LT 20.0] then <macro statement>
```

system variable for the current position of Axis 1 in the machine coordinate system

This fifth statement is evaluated based upon the position of X as 0.0, because the previous 4 lines of code have been preread and placed in the buffer, but have not yet been executed. The system variable for the current position of the X axis will not be updated until these lines of code have been executed, which is too late for the fifth line of code to evaluate as the programmer intended.

There is a workaround to this problem that avoids this problem. Simply program five "dummy" blocks of code before any macro statements involving system variables. Typically, we recommend the following for block delays:

```
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10
```

This only delays code execution by 50 milliseconds, but avoids any problems caused by a delay in system variable value updating.

CAUTION

IF YOU DO NOT PROGRAM IN BLOCK DELAYS BEFORE STATEMENTS INVOLVING SYSTEM VARIABLES, YOU WILL GET UNEXPECTED RESULTS IN THE EXECUTION OF YOUR MACRO PROGRAMS.

7.2 Single Block and Optional Skip

Macro functions are executed by the ServoWorks G-Code Parser. Therefore, the blocks that set macro variable values cannot be executed with the operation support function switch Optional Skip, or there will be an error. Nor can the operation support function switch Single Block be effective for blocks that set macro variable values.

Chapter 8: Macro Examples: Automatic Tool Change (ATC) with ServoWorks S-100T or the ServoWorks S-100M Series

8.1 Overview

The following are examples of macros for automatic tool change (ATC) functions for ServoWorks S-100T, ServoWorks S-100M, ServoWorks S-120M or ServoWorks S-140M.

NOTE: ATC achieved with a macro program is different than the GE Fanuc way. Fanuc supports the M06 code (Tool Change). Any time a tool change is required, an "M06T__" command is given inside a part program. This "M06T__" command activates either a macro program or a PLC sequence program, or both.

8.2 ATC – the ServoWorks CNC Way

For ServoWorks CNC products, ATC can be achieved without M06 codes, by using macros and PLC.

First, write a macro program or a subprogram for tool change. This macro or subprogram includes motion commands, M codes and T codes.

Then, in the part program, call that macro or subprogram each time you require a tool change. (Instead of just writing "M06T__" in the part program, as you would do with Fanuc.)

8.3 ATC Example #1: Simple ATC Function

An example part program and an example macro program follow:

```
#520=3
M98 P9006   ← Calls the tool change macro program
G04 P1500
#520=5
M98 P9006   ← Calls the tool change macro program
M02
```

Figure 8-1: Part Program That Calls the Macro Program O9006.dat

```
M50
M5
G04 P1000

(SUB 9006 -- Tool Change)
(X Tool Change Position)
#500 = -0.2294
 (Y Safe Position: -2.0093)
#501 = -2.0093
 (Y Tool Change Position: -0.3206)
#502 = -0.3206
(Z Tool Change Position)
#503 = -3.8072
(Tool Spacing)
#504 = 1.950
(General Feedrate)
#506 = 200.0
(Z Safe Position)
#507 = -2.000

G20

IF [#520 EQ #511] GOTO 100
IF [#520 EQ 0] GOTO 20
IF [#511 EQ 0] GOTO 10
N20
 (Current Tool)
#510 = #511
(Next Tool)
#511 = #520
#508 = #500 + [#510-1]*#504

G00 G80 G90 G70 G44
G04
G53 Z0.
G00 G53 X#508 Y#501
G00 G53 Z#503
G01 G53 Y#502 F#506

N10
M10
G04 P2000 (Open Collet)
G00 G53 Z0.

IF [#520 EQ 0] GOTO 100
(Current Tool)
#510 = #511
(Next Tool)
#511 = #520
#509 = #500 + [#511-1]*#504
G00 G53 X#509 Y#502
G00 G53 Z#507
G01 G53 Z#503 F#506
M11
G04 P2000 (Close Collet)
G01 G53 Y#501 F#506
G00 G53 Z0.
N100 (Error exit)
M51
G04 P1000
M99
%
```

Figure 8-2: Macro Program O9006.dat

8.4 ATC Example #2: ATC Using Customized G, M and T Codes

Following is an example of an ATC function that uses a customized G, M and T codes to trigger tool changes in the executing part program.

First, the macro function parameters are set as follows:

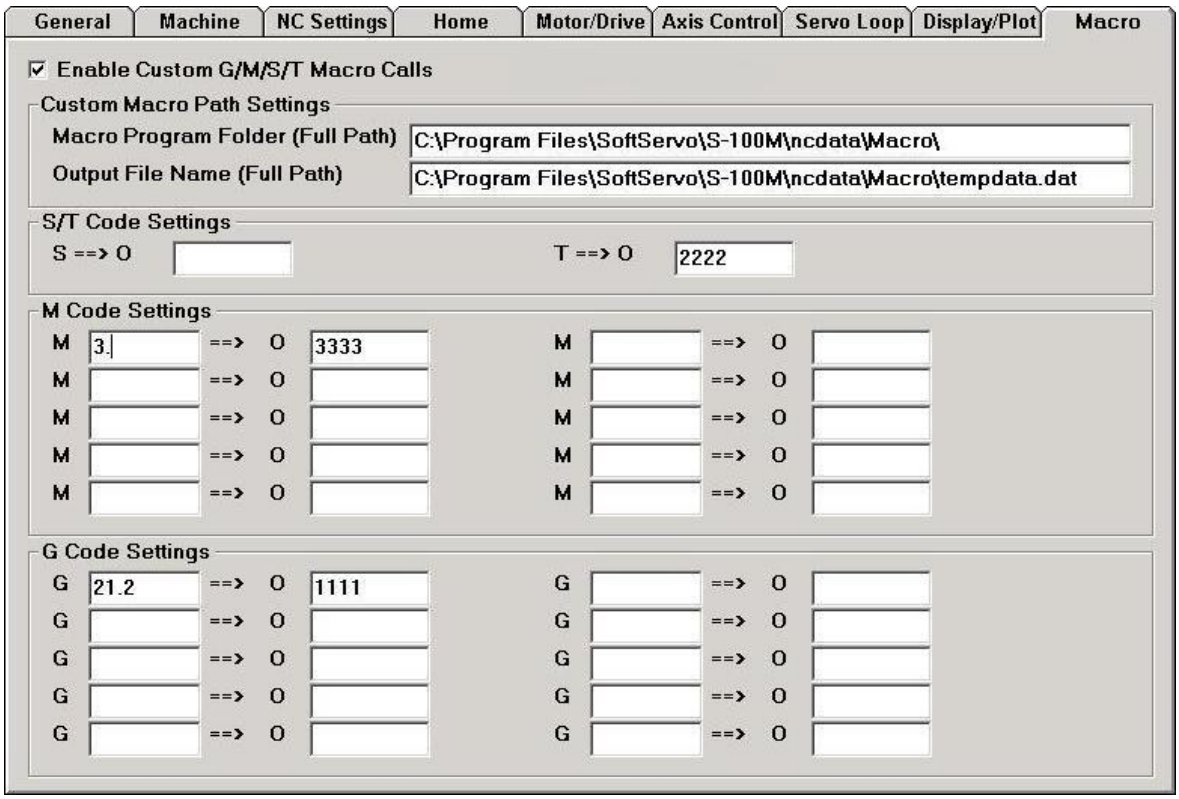


Figure 8-3: Example Macro Parameter Settings

Then, for each customized G, M, S or T code, you must write a customized macro program (.dat file) that resides in the macro program folder (C:\Program Files\SoftServo\S-100M\ncdata\Macro\ in this case). In this example, the customized macro programs are as follows:

```
G90                         (absolute programming)
G1 X-50.0 Y100.0 F2000      (linear interpolation command)
X-50.00 Y-200.0 F4000
M99                         (return to main program)
%
```

Figure 8-4: Example O1111.dat File

```
G90                         (absolute programming)
X100.0 Y-200.0
X0.0 Y0.0
M99                         (return to main program)
%
```

Figure 8-5: Example O2222.dat File

```
G90                         (absolute programming)
M03 S1000                   (spindle clockwise at 1000 RPMs)
M99                         (return to main program)
%
```

Figure 8-6: Example O3333.dat File

Write a part program file that uses these specialized S, T, M and G codes (G21.2, M03 and T20, in this example). One such example part program follows:

```
G53 X0.0 Y0.0
G92 X0.0 Y0.0
G021.2
X50.0 Y200.0
G1 X-50.0 Y100.0 F2000
X-50.005 Y-200.0 F4000
/G0
X-50.0 Y400.0
X100.0 Y100.0
X100.0 Y100.0
M03
G91G01 X50.0 Y0.002 F500.0
X-50.0 Y-0.001
G 0 X100.0 Y-0.001
G31 X-100.0 Y100.0 F2000.0
X100.0 Y100.0
G 4 P1000
G92 X50.0 Y20.0
G04 X2.0
G90X-100.0 Y400.0
X-50.0 Y100.0
T20
M1
X100.0 Y-200.0
\X-100.0 Y400.0
X0.0 Y0.0
M02
%
```

Figure 8-7: Example ATCTest.dat File

When the "Enable Custom G/M/S/T Macro Calls" parameter is set to "Enabled," the ServoWorks G-Code Parser searches the ATCTest.dat file before starting G code execution. It generates (creates) a temporary file (with the "Output File Name", in this case C:\Program Files\SoftServo\S-100M\ncdata\Macro\tempdata.dat), in which every special S, T, M or G code is replaced with a G65 simple macro call to the specified macro program, and each macro program is appended to the "tempdata.dat" file as a subroutine, as shown:

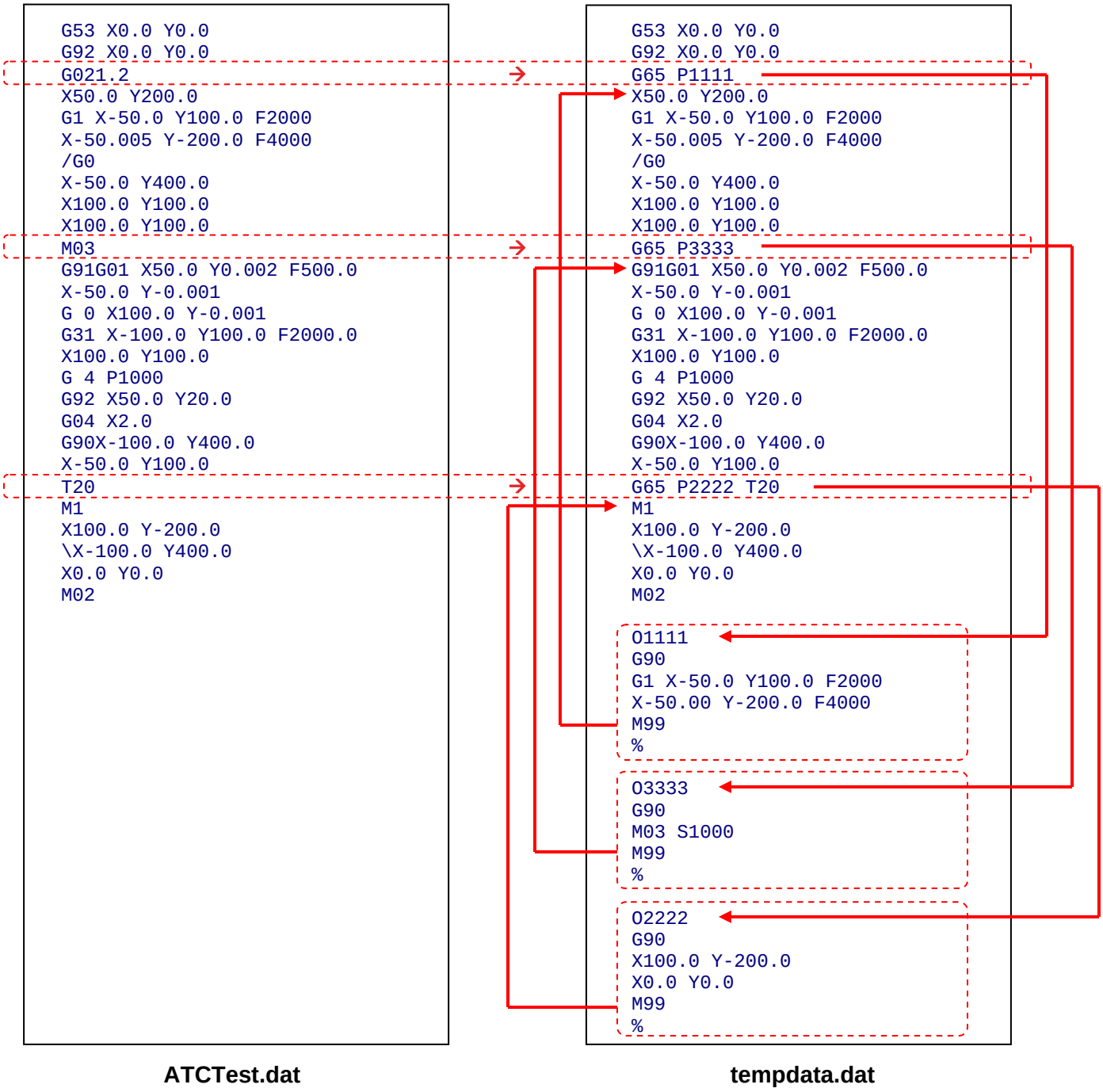


Figure 8-8: Example of ATC: Transformation of ATCTest.dat into the tempdata.dat File

The tempdata.dat file becomes the file actually executed by the ServoWorks G-Code Parser, NOT the ATCTest.dat file.

8.5 ATC Example #3: ATC for a Rotary Tool Changer Using PLC, Standard G Codes, and a Customized T Code

Following is another example of an ATC function that uses predefined M codes and a customized T code to trigger tool changes in the executing part program.

The rotary tool changer has 13 tool positions, and is shown in the following figure:

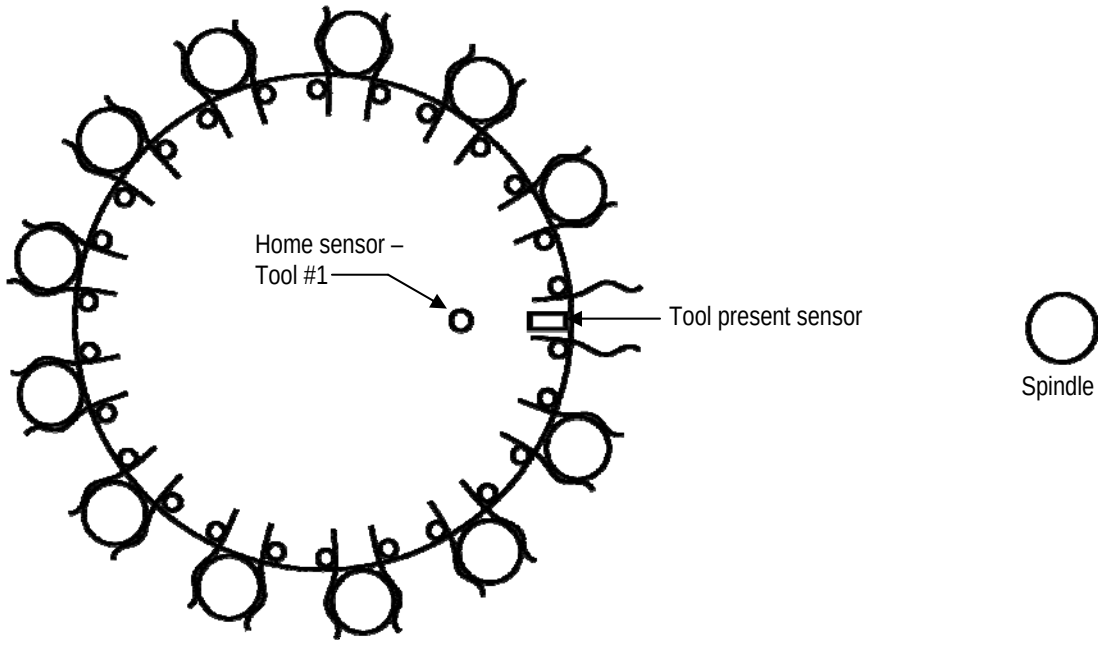


Figure 8-9: Rotary Tool Changer

NOTES:

1) BEFORE USING A T CODE TO INVOKE THE ATC MACRO PROGRAM, YOU MUST FIRST SET THE TOOL CHANGER TO ITS HOME POSITION USING M28, AND THEN USE MDI MODE TO SET #600 (A NUMBERED PERMANENT VARIABLE) TO 1 BY TYPING "#600=1" IN MDI MODE.

2) Variable #20 is assigned to address "T" (see Table 6-2). In this case, #20 is the tool number to be placed in the spindle. If "T7" was used to call the tool change macro program, then #20 = 7.

3) "M34" moves the carousel to the spindle.

4) "M35" moves the carousel back from the spindle.

When you want to invoke a tool change as part of a part program, just use "T" followed by the tool number (i.e. "T7"), and the O2004.dat tool change macro program will be invoked, and will pass the tool number to this macro program.

The tool change procedure that is programmed in the macro program for custom T codes (in file O2004.dat) is described in detail as follows:

1) Check to see if the called tool number is equal to the current tool number (does #20 equal #600?) – if it is, exit the macro program

2) Use the M12 code defined in the PLC sequence program to open the tool-changer cover

3) Check to make sure the tool-changer cover actually opened. Check the "tool change cover open sensor" wired to X2.5 – if it is 0 (open), continue, otherwise generate an alarm and exit the macro program. [More specifically, system variable #1005 is used in the macro program. #1005 corresponds to PLC address G54.5, and the PLC sequence program reads the value of X2.5, which is wired to the tool changer cover open sensor, and writes it to G54.5.]

4) Check the "tool present sensor" wired to X2.3 to see if there is a tool in the spindle – if it is 0 (no tool in the current holder), skip the tool return step, and continue to step #12

5) Move the spindle to the spindle safe position (above the workspace, to move over the tool-changer)

6) Move the spindle to the XY tool change position for tool return

7) Bring the spindle down to the tool change Z axis position

8) Move the tool-changer to return the current tool to the tool holder, moving at the low speed tool-changer feedrate

9) Open the collet using the M10 code defined in the PLC sequence program

10) Check the "open collet check" sensor wired to X2.4 to make sure the collet opened (is set to 1) – if it didn't, generate an alarm and exit the macro program

11) Grab the tool in the current holder by bringing the spindle down to the ready-to-approach position, and then retracting it back up to the safety position

12) Check to make sure the tool number isn't zero (in which case, having returned the current tool, go directly to step #24)

13) Calculate the difference between the current tool number and the called tool number, then calculate the distance (number of steps required to be travelled – sum of sections where each is 1/13 of a circle) to reach the called tool moving in each direction, to optimize the tool change operation

14) Check the "open collet check" sensor wired to X2.4 again, to make sure the collet is still open (is set to 1) – if it isn't, generate an alarm and exit the macro program

15) Rotate the carousel by the calculated angle to find the called tool and place it in the tool change position, using M24 (one full motor revolution to rotate the carousel to the next closest tool, 13 revolutions for full carousel revolution)

16) Move the spindle to the spindle safe position (above the workspace, to move over the tool-changer)

17) Move the spindle to the XY tool change position for tool return

18) Open the collet using the M10 code defined in the PLC sequence program

19) Check the "open collet check" sensor wired to X2.4 to make sure the collet opened – if it didn't, generate an alarm and exit the macro program

20) Grab the new tool from the current holder by bringing the spindle down to the ready-to-approach position (the grip position – the spindle clamps the tool with a spring), and then retracting it back up to the safety position

21) Close the collet and lock the tool using the M11 code defined in the PLC sequence program

22) Check the "open collet check" sensor wired to X2.4 to make sure the collet closed (is set to 0) – if it didn't, generate an alarm and exit the macro program

23) Save the current tool number as #600 (a numbered permanent variable)

24) Reverse the tool-changer to bring it to the tool holder safe position

25) Move the spindle to the spindle safe position (above the workspace, to move over the tool-changer)

26) Close the tool-changer cover using the M13 code defined in the PLC sequence program

27) Check to make sure the tool-changer cover actually closed. Check the "tool change cover open sensor" wired to X2.5 – if it is 1 (closed), continue, otherwise generate an alarm and exit the macro program.

28) Exit the tool change macro program, and return to the calling program.

The input wiring for this example is as follows:

ADDRESS	DESCRIPTION	NOTES
X2.0	Tool changer homing sensor	0: Not in position 1: In position
X2.1	Reached position switch for CW	0: Not in position 1: In position
X2.2	Reached position switch for CCW	0: Not in position 1: In position
X2.3	Tool present sensor	0: No tool in current holder 1: There is a tool in current holder
X2.4	Open collet check	0: Closed 1: Open
X2.5	Tool changer cover open sensor	0: Open 1: Closed
X2.6	Move carousel to spindle sensor	0: Not in position 1: In position
X2.7	Move carousel back from spindle sensor	0: Not in position 1: In position

Table 8-1: Input Wiring for ATC Example #3

The output wiring for this example is as follows:

ADDRESS	DESCRIPTION	NOTES
Y25.0	Tool changer rotate CW	0: Stop 1: Rotate
Y25.1	Tool changer rotate CCW	0: Stop 1: Rotate
Y25.3	Move carousel to spindle sensor	0: Stop 1: Move
Y25.4	Move carousel back from spindle sensor	0: Stop 1: Move

Table 8-2: Output Wiring for ATC Example #3

The macro function parameters must be set in Configuration Mode, Macro tab, as follows:

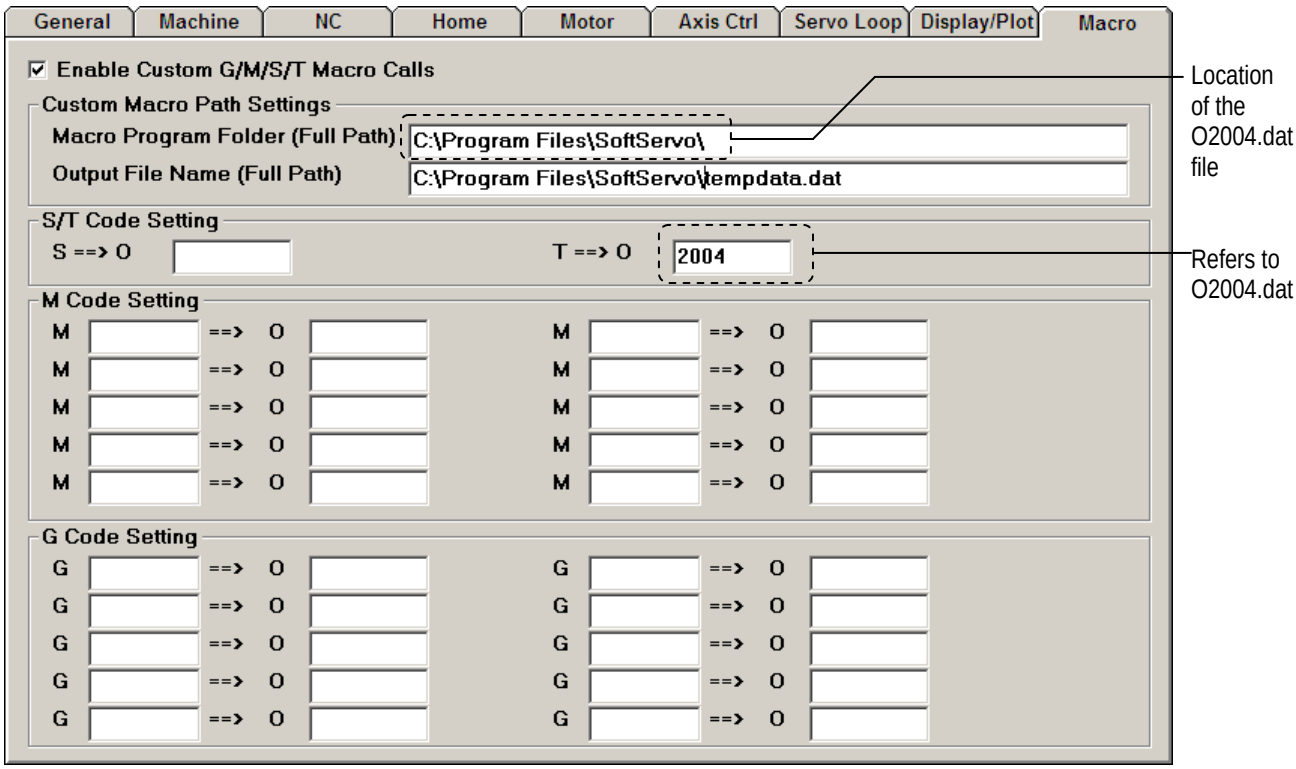


Figure 8-10: Example Macro Parameter Settings

Then, for the customized T code, the corresponding customized macro program (.dat file) for this example must resides in the macro program folder ('C:\Program Files\SoftServo' in this case). In this example, the customized macro program O2004.dat is as follows:

```
+  (Carousel type tool changer macro, 13 tools, 4 control inputs)
(O2004)
(M10 - open the collet)
(M11 - close the collet)
(M12 - open the tool-changer cover)
(M13 - close the tool-changer cover)
(#20 - tool to be placed in the spindle (T))

(Homing-After set the tool changer to the home position, set #600=1
 in MDI mode.)
(Participating I/O)
(Reached position switch for CW - input X2.0)
(Reached position switch for CCW - input X2.1)
(Tool present sensor - input X2.2)
(Open collet check - input X2.3)
(Tool changer cover open sensor X2.5)
(CW relay-output Y24.0)
(CCW relay-output Y24.1)
(M24 - one full motor revolution to rotate the carousel to the next
 closest tool, 13 revolutions for full carousel revolution)
(Single or bi-directional)

(#600)            (Current tool number, stored in HD permanently)

M05               (Safety spindle Stop)

#55=0.0           (Tool holder approach distance X axis)
#56=1.0           (Tool holder approach distance Y axis)
#57=1.0           (Spindle approach distance Z axis)
#50=1000          (M code time delay. Unit ms)
#9=10.0           (Low speed tool-changer feedrate)
#58=-1.5          (Spindle Z axis ready-to-approach position)
#59=0.5           (Spindle safety position - above the workspace to
                  move over the tool-changer)
#61=0.0           (Tool change X axis position)
#81=10.0          (Tool change Y axis position)

IF [#20 EQ #600] THEN M99
(If the called tool number is equal to the current tool number then
do nothing and exit the macro program)
                  (Message optionally)

M12               (open the tool-changer cover)
G04 P#50          (M-code time delay)
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10

IF [#1005 EQ 0] GOTO N10
(Check if the tool-changer cover is open - G54.5)
#1112=1           (Alarm message trigger F55.4)
M00
(Return the old tool)
```

Dummy delay blocks to ensure proper evaluation of the system variable (#1005) used in the following line of code – see *Section 7.1: Preread Function/Block Buffering – Problems and Workaround*

Figure 8-11: Example O2004.dat File (1 of 5)

```
N10
IF  [#1003 EQ 1] GOTO N40
(If there is no tool in the spindle, skip the tool return step,
otherwise continue)
IF [#1002 EQ 0] GOTO N20
(Check if the tool holder is empty for return - G54.2)
#1110=1           (Alarm message trigger F55.2)
M00

N20
G90G53 Z#59       (Safe spindle position)
G53 X#61 Y#81
(Place the spindle over the XY tool change position for tool return)
G90G53 Z[#58-#57]
(Place the spindle at the tool change Z axis position)
G91G01 X#55 Y#56 F#9
(Place the current tool in the tool holder)
M10               (Open the collet)
G04 P#50          (M-code time delay)
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10

IF [#1003 EQ 1] GOTO N30
(Collet open check G54.3)
#1111=1           (Alarm message trigger F55.3)
M00

N30
G91G01 Z#57       (Spindle up to a ready-to-approach position)
G90G53 Z#59       (Spindle up to the safety position)
(Grab the new tools)
```

Figure 8-12: Example O2004.dat File (2 of 5)

```
N40
IF [#20 EQ 0] GOTO N80
(Return the current tool but do not grab another)
#91=#20-#600
(Difference between the current tool number and the called tool
number)

IF [#91 GT 0]
(Calculating distances to reach for both directions to optimize )
     #90=#91
     (Number of steps to travel (sum of sections where each is 1/13
of a circle ))

     IF [#90 LT 7]
            #1100=0
            #1133=#90

     ELSE
            #1100=1
            #1133=13-#90

     ENDIF
ELSE
     #90=-#91
     (Number of steps to travel (sum of sections where each is 1/13
of a circle ))

     IF [#90 LT 7]
            #1100=1
            #1133=#90

     ELSE
            #1100=0
            #1133=13-#90

     ENDIF
ENDIF

IF [#1003 EQ 1]   GOTO N50
(Collet open check G54.3)
#1111=1           (Alarm message trigger F55.3)
M00
```

Figure 8-13: Example O2004.dat File (3 of 5)

```
N50
M24               (Rotate the carousel by given angle to find the
                  called tool and place it in the tool change
                  position)
G04 P#50
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10
G90G53 Z#59       (Safe spindle position)
G90G53 X#61 Y#81  (Spindle above the called tool position)
M10               (Open the collet)
G04 P#50
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10

IF [#1003 EQ 1]   GOTO N60
(Collet open check G54.3)
#1111=1           (Alarm message trigger F55.3)
M00

N60
G90G53 Z#58
(Place the spindle at the ready-to-approach Z axis position)
G91G01 Z-#57 F#9
(Shift the spindle down on a spindle to the grip position)
M11               (Close the collet)
G04 P#50
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10

IF [#1003 EQ 0] GOTO N70 (Collet close check G54.3)

#1111=1           (Alarm message trigger F55.3)
M00
```

Figure 8-14: Example O2004.dat File (4 of 5)

```
N70
#600=#20          (Save the current tool number)

N80
G91G01 X-#55 Y-#56(Reverse the tool holder)
G90G53 Z#59       (Spindle up to the safety position)
M13               (Close the tool-changer cover)
G04 P#50          (M-code time delay)
G04 P#50          (M-code time delay)
G04 P#50          (M-code time delay)
G04 P10
G04 P10
G04 P10
G04 P10
G04 P10

IF [#1005 EQ 1] GOTO N90
(Check if the tool-changer cover is closed - G54.4)

#1112=1           (Alarm message trigger F55.4)

M00

N90

M99
%
```

Figure 8-15: Example O2004.dat File (5 of 5)

The relevant part of the corresponding PLC sequence program that corresponds to the O2004.dat file is shown on the following pages. This sequence program defines M03 – M06, M08 – M17, M19 – M41.

```
// R3.0 - R6.7 for M code decoding
// M03 (Spindle CW) - Y24.0
RD            F10.0
AND           F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R3.3

// M04 (Spindle CCW) - Y24.1
RD.NOT        F10.0
AND.NOT       F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R3.4

// M05 (Spindle Stop - M03 or M04 OFF)
RD            F10.0
AND.NOT       F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R3.5

// M06 (Tool Change)
RD.NOT        F10.0
AND           F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R3.6

// M08 (Coolant On) - Y24.2
RD.NOT        F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.0
```

Figure 8-16: Example Sequence Program File (1 of 13)

```
// M09 (M08 OFF – Coolant Off)
RD            F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.1

// M10 - Y24.3
RD.NOT        F10.0
AND           F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.2

// M11 (M10 OFF)
RD            F10.0
AND           F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.3

// M12 - Y24.4
RD.NOT        F10.0
AND.NOT       F10.1
AND           F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.4

// M13 (M12 OFF)
RD            F10.0
AND.NOT       F10.1
AND           F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.5
```

Figure 8-17: Example Sequence Program File (2 of 13)

```
// M14 - Y24.5
RD.NOT        F10.0
AND           F10.1
AND           F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.6

// M15 (M14 OFF)
RD            F10.0
AND           F10.1
AND           F10.2
AND           F10.3
AND.NOT       F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R4.7

// M16 - Y24.6
RD.NOT        F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.0

// M17  (M16 OFF)
RD            F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.1

// M19 (Spindle Orientation On)
RD            F10.0
AND           F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.3
```

Figure 8-18: Example Sequence Program File (3 of 13)

```
// M20 (Spindle Orientation Off)
RD.NOT        F10.0
AND.NOT       F10.1
AND           F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.4

// M22 - Y24.7
RD.NOT        F10.0
AND           F10.1
AND           F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.6

// M23 (M22 OFF)
RD            F10.0
AND           F10.1
AND           F10.2
AND.NOT       F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R5.7


// M24 - Y25.0
RD.NOT        F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.0

// M25 (M24 OFF)
RD            F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.1
```

Figure 8-19: Example Sequence Program File (4 of 13)

```
// M26 - Y25.1
RD.NOT        F10.0
AND           F10.1
AND.NOT       F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.2

// M27 (M26 OFF)
RD            F10.0
AND           F10.1
AND.NOT       F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.3

// M28
RD.NOT        F10.0
AND.NOT       F10.1
AND           F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.4

// M29 (Spindle Rigid Mode On)
RD            F10.0
AND.NOT       F10.1
AND           F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.5

// M30 - Y25.2
RD.NOT        F10.0
AND           F10.1
AND           F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.6
```

Figure 8-20: Example Sequence Program File (5 of 13)

```
// M31 (M30 OFF)
RD            F10.0
AND           F10.1
AND           F10.2
AND           F10.3
AND           F10.4
AND.NOT       F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R6.7

// M32 - Y25.3
RD.NOT        F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.0

// M33 (M32 OFF)
RD            F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.1

// M34 - Y25.4
RD.NOT        F10.0
AND           F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.2

// M35 (M34 OFF)
RD            F10.0
AND           F10.1
AND.NOT       F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.3
```

Figure 8-21: Example Sequence Program File (6 of 13)

```
// M36 - Y25.5
RD.NOT        F10.0
AND.NOT       F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.4

// M37 (M36 OFF)
RD            F10.0
AND.NOT       F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.5

// M38 - Y25.6
RD.NOT        F10.0
AND           F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.6

// M39 (M38 OFF)
RD            F10.0
AND           F10.1
AND           F10.2
AND.NOT       F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R7.7

// M40 - Y25.7
RD.NOT        F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R8.0
```

Figure 8-22: Example Sequence Program File (7 of 13)

```
// M41 (M40 OFF)
RD            F10.0
AND.NOT       F10.1
AND.NOT       F10.2
AND           F10.3
AND.NOT       F10.4
AND           F10.5
AND.NOT       F10.6
AND.NOT       F10.7
WRT           R8.1

// Spindle stop
RD.NOT        F1.4
RD.STK        Y0.0   // DC-120/DC-150 output bit 0
AND.NOT       Y0.1
AND.NOT       Y0.2
OR.STK
WRT           Y0.0   // DC-120/DC-150 output bit 0

// Spindle CW
RD            F7.0
AND           R3.3
RD.STK        Y0.1
AND           F1.4
OR.STK
AND.NOT       R3.4
WRT           Y0.1

// Spindle CCW
RD            F7.0
AND           R3.4
RD.STK        Y0.2
AND           F1.4
OR.STK
AND.NOT       R3.3
WRT           Y0.2

// Spindle enable
RD            R3.3
OR            R3.4
RD.STK        R2.0
AND.NOT       R3.5
OR.STK
WRT           R2.0   // Spindle enable relay

// Enable spindle output with timer delay
RD            R2.0
TMR           2
WRT           G29.6

// Spindle Rigid Mode
RD            R6.5
WRT           G27.6
```

Figure 8-23: Example Sequence Program File (8 of 13)

```
// Spindle Orientation and C axis control
RD            R5.3
//RD.STK      G29.5  // Spindle orientation
RD.STK        G27.7  // C axis control
AND           F1.6
OR.STK
AND.NOT       R5.4
//WRT         G29.5  // Spindle orientation
WRT           G27.7  // C axis control

// Spindle Brake
RD.NOT        R2.0
TMR           3
WRT           Y1.3


//M code into physical output conversion

// (M03/04/M05)
RD            R3.3
RD.STK        Y24.0
AND.NOT       R3.4
AND.NOT       R3.5
OR.STK
WRT           Y24.0

RD            R3.4
RD.STK        Y24.1
AND.NOT       R3.3
AND.NOT       R3.5
OR.STK
WRT           Y24.1


// Coolant On/Off (M08/M09)
RD            R4.0
RD.STK        Y24.2
AND.NOT       R4.1
OR.STK
WRT           Y24.2

// (M10/M11)
RD            R4.2
RD.STK        Y24.3
AND.NOT       R4.3
OR.STK
WRT           Y24.3

// (M12/M13)
RD            R4.4
RD.STK        Y24.4
AND.NOT       R4.5
OR.STK
WRT           Y24.4
```

Figure 8-24: Example Sequence Program File (9 of 13)

```
// (M14/M15)
RD            R4.6
RD.STK        Y24.5
AND.NOT       R4.7
OR.STK
WRT           Y24.5

// (M16/M17)
RD            R5.0
RD.STK        Y24.6
AND.NOT       R5.1
OR.STK
WRT           Y24.6

// (M22/M23)
RD            R5.6
RD.STK        Y24.7
AND.NOT       R5.7
OR.STK
WRT           Y24.7

// (M24/M25)
RD.NOT        R200.0
AND           R06.0
AND.NOT       R6.1
WRT           R200.7
//ROTATE
RD            R200.7
AND           F54.0
WRT           Y25.0  //CW
RD            R200.7
AND.NOT       F54.0
WRT           Y25.1  //CCW

//M28 Home tool changer
RD            R200.5
OR            Y25.0
WRT           Y25.0

// (M30/M31)
RD            R6.6
RD.STK        Y25.2
AND.NOT       R6.7
OR.STK
WRT           Y25.2

// M34 MOVES CAROUSEL TO SPINDLE
RD            R201.2
WRT           Y25.3

// M35 MOVES CAROUSEL BACK FROM SPINDLE
RD            R201.3
WRT           Y25.4
```

Figure 8-25: Example Sequence Program File (10 of 13)

```
// (M36/M37)
RD          R7.4
RD.STK      Y25.5
AND.NOT     R7.5
OR.STK
WRT         Y25.5

// (M38/M39)
RD          R7.6
RD.STK      Y25.6
AND.NOT     R7.7
OR.STK
WRT         Y25.6

// (M40/M41)
RD          R8.0
RD.STK      Y25.7
AND.NOT     R8.1
OR.STK
WRT         Y25.7

// MFIN (M00, M01, M02, M03, M04, M05, M06, M08, M09, M10,
M11, M12, M13, M14, M15, M16, M17, M19, M20, M22, M23, M24,
M25, M26, M27, M30, M31, M32, M33, M34, M35, M36, M37, M38,
M39, M40, M41)

// M00, M01, M02, M30 decode
RD     F9.7
OR     F9.6
OR     F9.5
OR     F9.4
WRT    R0.0

//Custom M function decode
RD     R3.3
OR     R3.4
OR     R3.5
OR     R3.6
WRT    R0.1

RD     R4.0
OR     R4.1
OR     R4.2
OR     R4.3
OR     R4.4
OR     R4.5
OR     R4.6
OR     R4.7
WRT    R0.2

RD     R5.0
OR     R5.1
OR     R5.2
OR     R5.3
OR     R5.4
OR     R5.5
OR     R5.6
OR     R5.7
WRT    R0.3
```

Figure 8-26: Example Sequence Program File (11 of 13)

```
RD      R200.2
OR      R6.1
OR      R6.3
OR      R200.4
OR      R6.5
OR      R6.6
OR      R6.7
WRT     R0.4

RD      R7.0
OR      R7.1
OR      R201.0
OR      R201.1
OR      R7.4
OR      R7.5
OR      R7.6
OR      R7.7
WRT     R0.5

RD      R8.0
OR      R8.1
WRT     R0.6

//MFIN (by M strobe)
RD      R0.0
OR      R0.1
OR      R0.2
OR      R0.3
OR      R0.4
OR      R0.5
OR      R0.6
OR      R6.5
AND     F7.0
WRT     G5.0

// SFIN (By S Strobe)
RD      F7.2
WRT     G5.2

// TFIN (By T Strobe)
RD      F7.3
WRT     G5.3

//Tool change sensors
RD      X2.0   //Home sensor IN-0
AND     R06.4
WRT     R200.4
RD.NOT X2.0
AND     R06.4
WRT     R200.5

//Reached position switch CW and CCW sensors
RD            X2.1
AND           F54.0
RD.STK        X2.2
AND.NOT       F54.0
OR.STK
WRT           R200.3
```

Figure 8-27: Example Sequence Program File (12 of 13)

```
//Tool present sensor map to #1002
RD     X2.3
WRT    G54.2

//Open collet check map to #1003
RD     X2.4
WRT    G54.3

//Cover open check map to #1005
RD     X2.5
WRT    G54.5

//Motor to move carousel to spindle and back
RD     X2.6   //To sensor
AND    R07.2
WRT    R201.0
RD     X2.7   //Back sensor
AND    R07.3
WRT    R201.1
RD.NOT X2.6   //To sensor
AND    R07.2
WRT    R201.2
RD.NOT X2.7   //Back sensor
AND    R07.3
WRT    R201.3

//Counter
RD             R200.1
OR.NOT         R200.1
WRT            R200.1

RD.NOT         R200.1 //COUNT FROM 0
RD.NOT.STK     R200.1 //COUNT UP/DOWN
RD.STK         X2.5   //RST
RD.STK         R200.3
               //COUNT TRIGGER
SUB 55
F56
R202
WRT            R200.0

RD     R200.0
AND    R06.0
WRT    R200.2

//Alarm message
RD     F55.2
WRT    A0.2
RD     F55.3
WRT    A0.3
RD     F55.4
WRT    A0.4

%
```

Figure 8-28: Example Sequence Program File (13 of 13)

Index

T

U

V

W

X

Z