The NIST RS274/NGC Interpreter - Version 1

Thomas R. Kramer
Frederick Proctor
John L. Michaloski

Robot Systems Division
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce
Gaithersburg, Maryland 20899

April 28, 1994

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied.

Acknowledgements

Partial funding for the work described in this paper was provided to Catholic University by the National Institute of Standards and Technology under cooperative agreement Number 70NANB2H1213.

CONTENTS

1 Introduction

An interpreter has been written which reads numerical control code and produces calls to a set of canonical machining functions. The output of the interpreter can be used to drive a 3-axis machining center. This report describes version 1 of the interpreter.

1.1 Background

The Robot Systems Division (RSD) and the Automated Production Technology Division of the National Institute of Standards and Technology (NIST) are carrying out an Enhanced Machine Controller (EMC) project. One objective of the project is to demonstrate implementations of the Next Generation Controller (NGC) architecture. Another EMC objective is to demonstrate control of a Monarch VMC-75 3-axis machining center using an open-architecture control system. The EMC project has further objectives which we do not discuss here.

The NGC architecture has many independent parts, one of which is a specification for a numerical control code language for machining and turning centers. The specification is given in an August 24, 1992 report “RS274/NGC for the LOW END CONTROLLER - First Draft” [Allen Bradley] prepared by the Allen Bradley company. That language is called RS274/NGC in this report. The specification is an extension of a programming language for machine tools named RS274 which has been used for many years. The most recent prior version of RS274 is RS274-D, which was completed in 1979. It is described in the document “EIA Standard EIA-274-D” by the Electronic Industries Association [EIA].

As part of RSD assistance to the program which developed the NGC architecture, RSD prepared a report “NIST Support to the Next Generation Controller Program: 1991 Final Technical Report,” [Albus] containing a variety of suggestions. Appendix C to that report proposed three sets of commands for 3-Axis Machining, one set for each of three proposed hierarchical control levels. The suite proposed for the lowest (primitive) control level has been implemented by the EMC project as a set of functions in the C programming language which is known in the EMC project as the “canonical machining functions.” The EMC control system has been implemented so that the canonical machining functions can be used to control the Monarch VMC-75.

In the past two years, three translators were rapidly prototyped by one of the authors (Kramer) which read files of machining commands in a dialect of RS274-D and generate files of machining commands. All three of the translators read the version of RS274-D which was used by the GE2000 controller formerly attached to the Monarch VMC-75. The output of those two translators (one written in LISP, the other in C) is a file of canonical machining commands. For any given input file, the output files from those two translators are identical. The third translator writes output formatted like C functions implementing the NGC “Neutral Command Language,” yet another machine tool control language, which was developed by Martin Marietta and described in the “Controls Standardized Applications” report of the NGC program [Martin].

Beginning in July 1993, the authors have been developing a software system in the C language for reading machining commands in the RS274/NGC language and outputting canonical machining functions. Since the software can be used interactively, we are calling it an interpreter, rather than a translator. In the rest of this report, it is usually called “the RS274/NGC interpreter,” or simply “the interpreter.” This software builds on the C software developed for one the translators, but has been more carefully written.

1.2 Overview of RS274/NGC Code

1.2.1 Lines, Blocks, Commands, and Words

The RS274/NGC language is based on lines of code. Each line (also called a "block") may include commands to a machine tool to do several different things. A line is terminated by a carriage return or line feed. Lines of code may be collected in a file to make a program.

A line of code consists of an optional line number at the beginning followed by one or more "words," possibly interspersed with comments. A word normally consists of a letter followed by a number or an expression that can be evaluated to a number. A word may either give a command or provide a parameter to a command. For example, "G1 X3" is a valid line of code with two words. "G1" is a command meaning "move in a straight line at the programmed feed rate," and "X3" provides a parameter value (the value of X should be 3 at the end of the move) to the command. Most RS274/NGC commands start with either G or M (for miscellaneous). The words for these commands are called "G codes" and "M codes."

The term "word" is used extensively in [Allen Bradley] but is not clearly defined; [Allen Bradley, page 7] says: "Words in a part program consists of addresses and numeric values." In this report, a "word" is a single letter (possibly preceded by a comma) followed by a number or an expression.

In RS274/NGC, many a command stays active after being executed until some other command cancels it implicitly or explicitly. If coolant is turned on, it stays on until it is explicitly turned off. If a G1 command is given on one line, it will be executed again on the next line unless a command is given specifying a different motion (or some other command which implicitly cancels G1 is given).

1.2.2 Modal Groups

In RS274/NGC, G code and M code commands are either "modal" or "non-modal." Modal commands stay in effect until cancelled. Non-modal commands have effect only on the lines on which they occur. Modal commands are arranged in sets called "modal groups" [Allen Bradley, page 10 and page 105], and only one member of a modal group may be in force at any given time. For example, the G20 and G21 commands constitute G-group 6. G20 causes distances to be interpreted in inches while G21 causes millimeters to be used. In general, the modal groups contain commands for which it is logically impossible for two members to be in effect at the same time — like the inches vs. millimeters example just mentioned. A machine tool may be in many modes at the same time, with one mode from each modal group being in effect.

For several modal groups, [Allen Bradley] provides that when the machine is ready to accept commands, one member of the group must be in effect. Controller manufacturers must set default values for those modal groups. When a machine is turned on or otherwise re-initialized, the default values are automatically in effect.

Unfortunately, the handling of modal groups in RS274/NGC is inconsistent. For example, G-group 13 includes commands which are not logically mutually exclusive, and so does M-group 8. The motion commands in G-group 1 and G-group 9 combined are all logically mutually exclusive, but they are in separate groups. Several non-modal motion commands exist (G88.1 through G88.6) which could be modal and are logically mutually exclusive with the motion commands in G-group 1 and G-group 9. Commands which implicitly cancel a modal command are not necessarily from the same modal group as the cancelled command.

1.2.3 Observations

The specifications for RS274/NGC in [Allen Bradley] are sometimes vague or inconsistent. Where it is hard to tell what is intended in [Allen Bradley], we have picked what we believe is a sensible interpretation. A pervasive problem is that the order of execution of commands which may occur on the same line is not specified. It would be useful to write the definition of an allowable line of code in some production language (such as Wirth Syntax Notation) to make it precise. This has not been done, although we believe it is feasible.

Further details of the meaning of RS274/NGC code are given in later sections of this report.

1.3 Machine Models

In order to build an effective set of commands for any class of numerically controlled machines, it is necessary to know what machines in the class can do. A theoretical machine model is constructed to which all actual machines in the class must conform. RS274/NGC is aimed at two separate classes of machines: machining centers and turning centers. The language includes some commands applicable to both classes but there are separate sets of G codes for the two classes [Allen Bradley, sections 3.7 and 3.8]. The machining center model on which applicable parts of RS274/NGC are based is very similar to the machining center model on which the EMC canonical machining functions are based. The RS274/NGC model is more complex than the EMC model, but in most cases the two are consistent. In a few cases there are inconsistencies. Special provisions to handle these have been built into the interpreter.

The machine models are not discussed in detail in this report, but the places where they differ will be discussed.

1.4 Canonical Machining Functions

The EMC canonical machining functions called by the interpreter are given in Table 1 and are described in more detail in Appendix B of this report. Appendix C of [Albus] lists additional canonical machining functions which are not used by the current version of the interpreter but may be in future versions. The names of the functions explain roughly what they do. More details are given in Appendix B.

Two sets of definitions for the machining functions have been written, and either set can be linked into the interpreter. The first set is used for direct control of the machining center. Executing a function from this set causes one or more lower-level commands to be sent to the machine's actuator controllers. The second set is used for testing or for writing a command file that can be used later. Executing a function from the second set causes a line of text containing the command to be written to standard output or to a file.

Representation
SET_PROGRAM_ORIGIN (double x, double y, double z)
USE_ABSOLUTE_ORIGIN ()
USE_LENGTH_UNITS (char * inunit)
USE_PROGRAM_ORIGIN ()

Free Space Motion
SET_TRAVERSE_RATE (double rate)
STRAIGHT_TRAVERSE (double x, double y, double z)

Machining Attributes
SELECT_PLANE (char * inplane)
SET_FEED_RATE (double rate)
STOP_CUTTER_RADIUS_COMPENSATION () { }

Machining Functions
ARC_FEED (double first_axis, double second_axis, double rotation, double axis_end_point)
STRAIGHT_FEED (double x, double y, double z, double probe)

Spindle Functions
SET_SPINDLE_SPEED (double r)
SPINDLE_RETRACT_TRAVERSE ()
START_SPINDLE_CLOCKWISE ()
START_SPINDLE_COUNTERCLOCKWISE ()
STOP_SPINDLE_TURNING ()
USE_NO_SPINDLE_FORCE () { }

Tool Functions
CHANGE_TOOL (int i)
USE_MODIFIED_TOOL_LENGTH_OFFSETS (int r)
USE_NORMAL_TOOL_LENGTH_OFFSETS ()
USE_NO_TOOL_LENGTH_OFFSETS ()

Miscellaneous Functions
COMMENT (char * s)
FLOOD_OFF ()
FLOOD_ON ()
MIST_OFF ()
MIST_ON ()

Program Functions
OPTIONAL_PROGRAM_STOP ()
PROGRAM_END ()
PROGRAM_STOP ()

Table 1. Canonical Machining Functions Called By Interpreter

All functions return nothing. Arguments are shown in parentheses in ANSI C format.

2 Overview of the Interpreter

2.1 Major Characteristics

2.1.1 Modes of Use

The interpreter has two output modes and two input modes. As described in the preceding section, output consists of executing commands or writing text, depending on which set of canonical machining functions has been linked into the interpreter. The input modes are either interactive or batch mode. In the interactive mode, the user types lines of RS274/NGC code at the keyboard. In the batch mode, the interpreter reads lines of code from a file. Thus, there are four possible combinations of input and output:

keyboard input and printed output,
keyboard input and command output,
file input and printed output,
file input and command output.

When using file input and printed output, the output may be redirected to a file for later use. It is not useful to redirect printed output when using the keyboard because the output is full of prompts as well as command text.

2.1.2 How it Runs

Once initialized, the interpreter runs in a loop in which it repeatedly reads another line of RS274/NGC code and calls one or more canonical machining functions.

The interpretation is done by reading the line, building an internal representation of the meaning of the entire line, and then calling the appropriate canonical function(s) to carry out what the line says to do.

2.1.3 Speed

Using an input test file for machining a semicircular arc back and forth 1000 times in a row (for a total of 2000 lines), running on a SUN SPARCstation 2, the interpreter wrote an output file in 5 seconds.

The speed of the interpreter is not very important when it is being used to write printed output, since the output is not being used in real time. When the interpreter is being used in real time, there are relatively few situations in which even the slowest modern computer would be challenged. Interpreting in real time a file consisting of many consecutive short arcs (say 0.1 millimeter long each) is probably the toughest realistic challenge. At 1000 millimeters per minute feed rate (a realistic value), a 2000-line file should run in 12 seconds. The test results just cited indicate that the interpreter running on a SPARCstation 2 is somewhat faster than that. Higher feed rates might cause the interpreter to be unable to keep up, with unknown effects.

2.2 Start-up

When the interpreter starts up, before accepting any input, it executes several non-motion commands to bring the machine tool to a known state. Then it executes a straight_traverse command to bring the spindle of the machine tool to a known position. If keyboard input is being used, the user is asked to give the OK before these commands are given. The initialization commands bring the machine to the following state:

spindle	not turning
spindle speed	0
traverse rate	100
feed rate	0
flood coolant	off
mist coolant	off
cutter radius compensation	off
effective coordinate system	absolute
program origin	point (0, 0, 0)
length units	inches
spindle force	off
selected plane	XY plane
tool length offset mode	normal

2.3 Error Handling

The interpreter performs a great deal of checking as it runs. If an error is found during any check, the interpreter prints one or more messages describing the error. Behavior after printing an error message depends upon the input mode. If keyboard input is being used, the interpreter does not try to execute any part of the line which contained the error, but it does continue to run and prompts the user for another line of input in the usual way — the idea being that the user will think about the error and take appropriate action. If file input is being used, the line on which the error occurred is not executed, and the interpreter exits — the idea being that perhaps no one is monitoring the interpreter and continuing after an error occurs is likely to be a waste of time (workpiece already ruined), dangerous, or both.

The error messages which may be printed by the interpreter are listed in Appendix D. They are self-explanatory.

Further details of error handling are given in Appendix A.

2.4 Exiting

When using keyboard input, the interpreter exits if it reads a line with an M2 (program exit) command on it. The rest of the line is executed before the interpreter exits.

When using file input, the interpreter exits when a line with an M2 command is read (also after executing the rest of the line) or when the end of the input file is reached, whichever comes first.

As stated earlier, when using file input, the interpreter will also exit if a line is read that causes an error.

3 Building an Executable

On a SUN SPARCstation 2, an executable file for the interpreter may be built from source code in less than a minute, as described below. The same procedure should work on any computer running a Unix operating system and having the standard C libraries. On computers running other operating systems, compilation should be similarly easy, provided the standard C libraries are available.

To make an executable, four source code files must be placed in the same directory along with the Makefile shown in Table 2 below. The source code files are:

canon_defs.c
canon_defs.h
ngc_to_prim.c
ngc_to_prim.h

The first two files are the function definitions and header file for the canonical machining functions. The second two are the function definitions and header file for the interpreter. As mentioned earlier, two different versions of canon_defs.c are available, one which issues commands to lower level machine controllers, and one which prints function calls.

An executable file named “ngc_to_prim” is built in the same directory by giving the command: **make ngc_to_prim**.

In the Makefile, we are using the Gnu C compiler, “gcc.” Any other ANSI-compliant C compiler may be substituted for gcc.

```
canon_defs.o: canon_defs.c
        gcc -c -g -O canon_defs.c
ngc_to_prim.o: ngc_to_prim.h ngc_to_prim.c canon_defs.h
        gcc -c -g -O ngc_to_prim.c
ngc_to_prim: ngc_to_prim.o canon_defs.o
        gcc -o ngc_to_prim ngc_to_prim.o canon_defs.o -lm
```

Table 2. Makefile for Interpreter

4 Using the Interpreter

4.1 Invoking the Interpreter

As mentioned earlier, the interpreter may be used with either keyboard input or file input.

The interpreter is invoked with keyboard input by giving the command:
ngc_to_prim

The interpreter is invoked with file input by giving a command of the form:
ngc_to_prim *input_filename*
where input_filename is the name of the input file. With this invocation, normal printed output from the interpreter (everything but error messages) appears on stdout. Printed output may be usefully redirected to an output file by giving a command of the form:
ngc_to_prim *input_filename* > *output_filename*

4.2 Keyboard User Interface

The interpreter has a simple line-based user interface for when it is used with keyboard input. The normal pattern of use is a read-execute cycle with four steps:

1. The interpreter prints the prompt **READ =>**

2. The user enters a line of RS274/NGC code at the keyboard and hits the carriage return button.

3. The interpreter reads the line, checks the line, and prints the prompt **EXEC <-**

4. The user enters a semicolon and hits the carriage return button, and the line is interpreted.

Step 4 has been included in the interface because the consequences of giving an incorrect command can be disastrous. If anything but a semicolon is entered in that step, the line is not interpreted. This will protect the user who accidentally hits the return button during step 2 and will give the user a second chance to think about each command.

The user interface provides no capability to edit ahead, undo, or anything else involving more than the current line.

A transcript of a short session with the interpreter using keyboard input is shown in Appendix C.

5 Capabilities of the Interpreter

The interpreter implements only a portion of RS274/NGC. The first version includes the following capabilities.

5.1 Input Letters

Allowable letters for starting words are: F, G, H, M, N, R, S, T, X, Y, Z

The meanings of the letters are as given in [Allen Bradley, pages 8 - 11 and elsewhere]:

F feedrate
G general function (see below)
H tool length offset
M miscellaneous function (see below)
N line number
R arc radius
S spindle speed
T tool selection
X x-axis of machine
Y y-axis of machine
Z z-axis of machine

5.2 Input General Functions

Interpretation of the following general functions (G words) has been implemented. Unless noted otherwise in this document, implementation is as described in [Allen Bradley]:

G0 rapid positioning
G1 linear interpolation
G2 circular/helical interpolation (clockwise)
G3 circular/helical interpolation (counterclockwise)
G10 coordinate system origin setting
G17 xy plane selection
G18 xz plane selection
G19 yz plane selection

G20 inch system selection
G21 millimeter system selection
G43 tool length offset (plus)
G49 cancel tool length offset
G54 use absolute origin
G55 use program origin
G90 absolute mode
G94 feed per minute mode

5.3 Input Miscellaneous Functions

The following miscellaneous functions (M words) are implemented as described in [Allen Bradley]:

M0 program stop
M1 optional program stop
M2 program end
M3 turn spindle clockwise
M4 turn spindle counterclockwise
M5 stop spindle turning
M6 tool change
M7 mist coolant on
M8 flood coolant on
M9 mist and flood coolant off

5.4 Expressions

Expressions [Allen Bradley, pages 70 - 73] involving numbers and four binary mathematical operators (plus, minus, times, divided-by) [Allen Bradley, page 71] have been implemented, including using brackets to indicate precedence of operations. Binary logical operators, system parameters, and unary operators [Allen Bradley, page 72] have not been implemented. At any given level of evaluation, if there is more than one binary operation, the operations are performed left to right; this differs from the specification of [Allen Bradley, page 71], which has two precedence classes for binary operators. For example, [1 + 2 * 3] evaluates to 9 in the interpreter (it becomes 3 * 3 at the first step in the evaluation) rather than to 7 (which it would be if multiplication took precedence over addition).

5.5 Comments

The interpreter will interpret comments. Details are given in Section 6.7.

6 INPUT

In general, allowable inputs are as described in [Allen Bradley]. That document references [EIA], the standard for RS274-D. Where the two documents conflict, the prescriptions of [Allen Bradley] are used. Where [Allen Bradley] is silent, but [EIA] has something to say, [EIA] is used.

6.1 Word Repeats

[Allen Bradley] does not specify explicitly whether two words starting with the same letter can occur in the same line, but portions of [Allen Bradley] (section 3.5.1.3 on page 73, for example)

imply clearly that this is legal. [EIA] has an explicit statement (item 3.12, page 5) that "words (apparently meaning words with the same initial letter) shall not be repeated in a block." We will assume that [EIA] does not apply. The interpreter uses the following rules, which were the ones used by the GE2000 controller for the Monarch VMC-75 and are probably similar to what is intended by the authors of [Allen Bradley]:

- A line may have one or two G words. Two G words from the same modal group may not appear on the same line.
- A line may have up to three M words. Two M words from the same modal group may not appear on the same line.
- For all other legal letters, a line may have only one word beginning with that letter.

6.2 Word order

[Allen Bradley] does not specify word order explicitly or implicitly. [EIA, pages 5, 6] puts limitations on word order. The interpreter uses the following rules:

- If there is an N word (a line number), it must be first.
- Words starting with other letters may occur in any order.

6.3 Measurement Units

6.3.1 Linear units

[Allen Bradley, page 21] specifies that either "inch" or "metric" units may be used by programming G20 for inch or G21 for metric. The default is up to the system builder.

[Allen Bradley] uses the term "metric" frequently (pages 5, 21, for example). The use of the term implies clearly that in many cases it is meant to denote a linear measurement. Where "metric" is used as a linear measurement, [Allen Bradley] does not say what metric unit is intended. [EIA, page 3] specifies millimeters, so the interpreter uses millimeters.

6.3.2 Angular units

[Allen Bradley, pages 18, 19, 34] specifies using degrees for measuring angles, so the interpreter assumes angular dimensions in the input are given in degrees.

6.4 Space Characters

[Allen Bradley] says nothing about space characters. [EIA, page 6, last line] allows spaces anywhere and provides that they should be "ignored by control." The interpreter allows spaces anywhere on a line of code and behaves the same as it would if the spaces were not there. This makes some strange-looking input legal ("g0x +0. 12 34y 7" is equivalent to "g0 x+0.1234 y7", for example).

6.5 Numbers

[Allen Bradley] is not clear regarding what a valid number is. On page 4 it says "The programming resolution of the axis word formats word length is 8." The most likely interpretation of this is that a word representing an axis motion can have at most eight characters (presumably including one letter at the beginning and a decimal point in the middle), providing room for six digits. The table on pages 8 and 9 provides for up to 14 digits, but a sentence at the bottom of page 7 says the table may not be valid, and "the control allows a maximum of 8 total digits."

[Allen Bradley, page 4] says “Decimal Point Programming of Axis motion will be the standard.” It does not say what Decimal Point Programming is, however. [EIA, page 3] does describe decimal point programming. It allows all unnecessary zeros to be included or suppressed and decimal points to be omitted if whole numbers are used.

The interpreter uses the following rules:

- A digit is a single character between 0 and 9.
- A number consists of (i) an optional plus or minus sign, followed by (ii) zero to many digits, followed, possibly, by (iii) one decimal point, followed by (iv) zero to many digits - provided that there is at least one digit somewhere in the number.
- There are two kinds of numbers: integers and decimals. An integer does not have a decimal point in it; a decimal does.
- Some numbers must always be integers, others may be either decimals or integers.
- Numbers may have any number of digits, subject to the limitation on line length.
- A number with no sign as the first character is assumed to be positive.

6.6 Case Sensitivity

[Allen Bradley] and [EIA] do not explicitly discuss character case. [EIA, page 1] incorporates by reference “the character code set of EIA RS-358 as the designated code set for Numerical Control,” which has not been examined. The interpreter assumes input is case insensitive, i.e., any letter may be in upper or lower case without changing the meaning of a line.

6.7 Comments

[Allen Bradley, page 5] prescribes using parentheses to give comments. It does not specify whether nested comments are allowed or whether a line can end in the middle of a comment without being closed by a right parenthesis. The interpreter assumes comments can occur anywhere in a line but may not contain left parentheses and must be terminated by a right parenthesis before the end of the line, or the line is invalid and will not be executed.

6.8 Line Numbers

[Allen Bradley, page 11] provides for sequence numbers (i.e., line numbers). Line numbers are optional. For the interpreter, if a line number is used, it must be an N followed by one or more digits. Line numbers do not have to be used in any particular order, and they may be repeated.

6.9 Programs

[Allen Bradley, pages 5, 6, 7] discusses “main programs” and “subprograms.” It is provided that the first word on the first line of a file containing a main program or a subprogram should be an O word. This word is to be considered to be the name of the file for calling it from another program. The last line of a file for a main program should have M2, M30, or M99 on it.

[EIA] does not discuss programs, so it has nothing to add to the above.

The interpreter does not read O words, does not deal with subprograms, and does not recognize M30 or M99. The interpreter does not have the concept of a program. However, the interpreter will exit when a line with M2 on it is encountered, after interpreting the rest of the line.

6.10 Blank Lines

Blank lines are allowed in the input by the interpreter. They are ignored.

7 OUTPUT

7.1 Functions Called

The interpreter calls canonical machining functions, which are as described in Appendix B. If the version of the canonical machining functions which prints functions calls is linked into the interpreter, the printing goes to stdout. Output which goes to stdout may, of course, be redirected to a file.

7.2 Other Output

Other interpreter output consists of prompts to stdout (if keyboard input is being used) and error messages to stderr.

8 Limitations of the Interpreter

The interpreter implements the parts of RS274/NGC which are expected to be most heavily used, but this includes only 10% to 20% of the language. This section calls out the major limitations of the interpreter.

8.1 Many G Codes and M Codes Not Implemented

The interpreter handles 16 G codes; [Allen Bradley] defines just over 100 G codes. The interpreter handles 10 of the 20 M codes defined in [Allen Bradley].

8.2 No Commands for Transfer of Control Implemented

The commands for transfer of control specified in [Allen Bradley, pages 74 - 76] are not implemented. This includes: conditional operators and GOTO, IF-GOTO, and WHILE-DO-END commands.

8.3 Expressions

Binary logical operators, system parameters, and unary operators [Allen Bradley, pages 71 - 73] have not been implemented. Classes of binary operators with different precedence levels have not been implemented.

8.4 Parameters

Parameter setting and use is not implemented.

8.5 Arcs

Only the forms of arc commands (G2 and G3) that use the radius, R, are implemented. The forms that use the center (which require I, J, and K) have not been implemented.

8.6 Input Not Saved

The interpreter does not save input lines. This means that RS274/NGC programs cannot be written using the interpreter, which is of little importance. It also means that there is no record of what the user did, which is more important.

8.7 Machine Model Limited

The machine model of the canonical machining functions is simpler than that of RS274/NGC. As described in Appendix A, this makes some RS274/NGC commands, particularly those regarding coordinate systems and tool length offsets, hard to interpret.

The machine model of the canonical machining functions does not provide for getting any information out of the machining center, such as the current position or the value of a tool length offset. The lack of tool length offset values causes the interpreter to not know what the z-value of the current position is whenever a tool is changed and tool length offsets are being used.

This version of the interpreter takes the view that the interpreter is a temporary third party between the RS274/NGC code and the machine tool. The interpreter is not expected to remember anything about the settings of the machine tool or about previous sessions between the times it is used. It would be feasible in future versions of the interpreter, and probably desirable, for the interpreter to be able to run in a mode in which it looks like part of the machining center to the user who wants to run RS274/NGC code. By maintaining files of information about machine settings between sessions, the interpreter could make the machining center conform exactly to the RS274/NGC machine model, and all applicable commands of the RS274/NGC language could be implemented.

References

[Albus] Albus, James S; et al; NIST *Support to the Next Generation Controller Program: 1991 Final Technical Report*; NISTIR 4888; National Institute of Standards and Technology, Gaithersburg, MD; July 1992

[Allen Bradley] Allen Bradley; *RS274/NGC for the Low End Controller*; First Draft; Allen Bradley; August 1992

[EIA] Electronic Industries Association; *EIA Standard EIA-274-D Interchangeable Variable Block Data Format for Positioning, Contouring, and Contouring/ Positioning Numerically Controlled Machines*; Electronic Industries Association; Washington, DC; February 1979

[Martin] Martin Marietta; *Controls Standardized Application (CSA)*; Draft Volume V of Next Generation Workstation/Machine Controller (NGC) Specification for an Open System Architecture Standard (SOSAS); Martin Marietta Document No. NGC-0001-14-000-CSA; March 1992

Appendix A Software and Functional Details

This appendix describes the software for the interpreter and additional functional details of the interpreter. The appendix is intended for users concerned with fine details and for programmers who might want to modify the software or simply understand it.

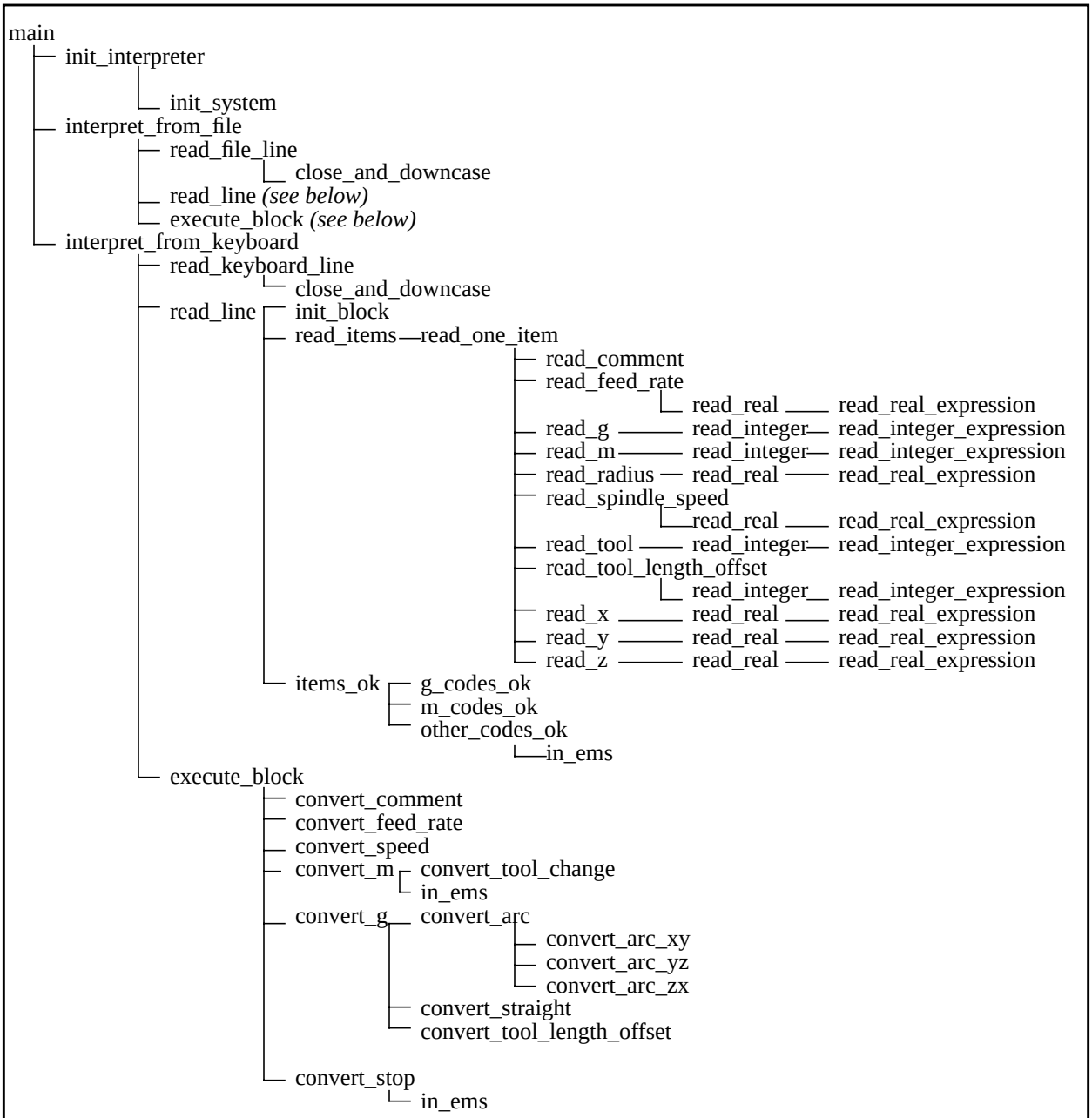

Figure 1. Interpreter Function Call Hierarchy

Not shown: read_real_expression calls read_real; read_integer_expression calls read_integer.

A.1 Software Introduction

The interpreter is written in ANSI C. The necessary program files are ngc_to_prim.c, ngc_to_prim.h, canon_defs.c, and canon_defs.h. The function call hierarchy is shown in Figure 1.

A.2 Error Handling and Exiting

Error handling and exiting are handled together in the software by returning a status value of OK, EXIT, or ERROR from each function where a reason for exiting may be detected or where there is likelihood of error. One or more appropriate error messages are printed to stderr for each error.

The interpreter exits only from main. Control is passed back up through the function call hierarchy to main before the exit. When reading from the keyboard, an error will not cause the interpreter to exit. When reading from a file, any error will cause interpretation to stop at the line that caused the error, and the interpreter will exit.

Since returned values are usually used as just described to handle the possibility of errors, an alternative method of passing calculated values is required. In general, if function A needs a value for variable V calculated by function B, this is handled by passing a pointer to V from A to B, and B calculates and sets V.

A.3 Cyclic Operation

Once the interpreter is initialized, the software continually repeats a read-store-check-execute cycle until stopped by an error or exit command.

A.3.1 Read

The software reads lines of RS274/NGC code one at a time. First the line is read into a buffer. Next, any line number is removed from the front of the buffer (everything else is shifted to the front of the buffer), any spaces not in comments are removed, and any upper case letters not in comments are changed to lower case letters.

Then the buffer and a counter holding the index of the next character to be read are handed around among a lot of functions named read_*XXXX*. All such functions read characters from the buffer using the counter. They all reset the counter to point at the character in the buffer following the last one used by the function. The first character read by most of these functions is expected to be a member of some set of characters (often a specific single character), and each function checks the first character.

A.3.2 Store

For each line of code a “block” structure is built. Every block has a slot for every potential piece of information on the line. Each time a useful piece of information has been extracted from the buffer, the information is put into the block.

A.3.3 Check

The read_*XXXX* functions do a lot of error detection, but they only look for errors which would cause reading or storing to fail. After reading and storing is complete, more checking functions (g_codes_ok, m_codes_ok, other_codes_ok) are run. These functions look for logical errors, such as more than one command in a block coming from the same modal group.

A.3.4 Execute

Once a block is built and checked, the block is executed by some function named convert_XXXX called by the execute_block function. The order of execution is described in Appendix A.8.

A.4 Tool Change

[Allen Bradley, pages 108, 109] provides four methods for changing a tool, but there are errors in the text that make it hard to tell what is intended. In the interpreter, both M6 and a T word are required on the same line to change a tool. It is an error to use only one of the two on a line. The interpreter output makes a tool change occur as follows:

- If there is an F word on the line, the feed rate is reset.
- If there is an S word on the line, the spindle speed is reset.
- The spindle stops turning.
- The spindle retracts fully at positioning speed.
- If there is a tool in the spindle, it is returned to its correct slot on the carrousel.
- The tool designated by the T word is removed from the carrousel and inserted in the spindle.
- If there is an M3 or M4 on the line, the spindle starts turning again. If not, the spindle stays stopped.
- Other commands on the line, such as axis motion, coolant on or off, tool length compensation, etc., occur after the above in a suitable order.

A.5 Milling Arcs

In RS274/NGC code [Allen Bradley, pages 19, 20, 27, 28] a circular or helical arc is specified using either G2 (clockwise arc) or G3 (counterclockwise arc). The axis of the circle or helix must be parallel to the x-, y-, or z-axis of the machine coordinate system. The axis (or, equivalently, the plane perpendicular to the axis) is selected with G17 (z-axis, xy-plane), G18 (y-axis, xz-plane), or G19 (x-axis, yz-plane). If the arc is circular, it lies in the selected plane.

In RS274/NGC, two formats are allowed for specifying an arc. We will call these the center format and the radius format. The interpreter only interprets the radius format with a positive radius. In the radius format the coordinates of the end point of the arc in the selected plane are specified along with the radius of the arc. A positive radius indicates that the arc turns through 180 degrees or less. If the arc is helical, the value of the end point of the arc on the coordinate axis parallel to the axis of the helix is also specified.

Here is an example of an RS274/NGC command to mill an arc: G17 G2 x 10 y 15 r 20 z 5

That means to make a clockwise (as viewed from the positive z-axis) circular or helical arc whose axis is parallel to the z-axis, ending where x=10, y=15, and z=5, with a radius of 20. If the starting value of z is 5, this is a circular arc; otherwise it is a helical arc.

The interpreter converts arcs to a center format (different from the RS274/NGC center format) as described in Appendix B.4.1.

A.6 Coordinate Systems

The handling of coordinate systems by the interpreter involves the RS274/NGC commands G10, G54, and G55.

The canonical machining functions model of coordinate systems is:

1. There are two coordinate systems: absolute and program.
2. The user select one of the two by giving either the USE_ABSOLUTE_ORIGIN command or the USE_PROGRAM_ORIGIN command.
3. The user can set the offsets of the program coordinate system with the SET_PROGRAM_ORIGIN command.
4. The absolute coordinate system is the default system.

The RS274/NGC model of coordinate systems, as given in section 3.2 of the manual [Allen Bradley, pages 13 - 19] is:

1. There are ten coordinate systems: absolute and 9 program.
2. The user can select one of the 9 program coordinate systems by giving a G54, G55, G56, G57, G58, G59, G59.1, G59.2, or G59.3 command (which select systems 1 - 9 respectively). The user cannot select the absolute coordinate system.
3. The user can set the offsets of the 9 program coordinate systems by giving a command of the form G10 L2 P*n* X*r* Y*s* Z*t*, where *n* is between 1 and 9. *r, s*, and *t* are real numbers giving the absolute coordinates of the origin of the program coordinate system.
4. The first one of the 9 program coordinate systems is the default.

The approach used by the interpreter mates the canonical view of coordinate systems with the RS274/NGC view by setting the offsets for the RS274/NGC default coordinate system to zero (so that it is the machine coordinate system) and by using the second of the 9 coordinate systems as the one alternative program coordinate system. This is implemented as follows.

A G54 command is interpreted into USE_ABSOLUTE_ORIGIN. G55 is interpreted into USE_PROGRAM_ORIGIN.

G10 is used to alter the second program coordinate system as described in [Allen Bradley, pages 13, 14], except that in the interpreter, G10 means what “G10 L2 P2” means in [Allen Bradley]. This version of the interpreter cannot read L and P words, so they cannot be used explicitly. Thus, G10 can be used only to modify the coordinate table entries corresponding to G55. The coordinate table entries corresponding to G54 (the default coordinate system) are set to zero and cannot be reset, so that the default coordinate system is the machine coordinate system.

A.7 Tool Length Offsets

The canonical machining commands model of using tool length offsets differs appreciably from the RS274/NGC model, as follows. The user will need to be aware of this to get tool length offsets to behave as expected.

Canonical Model - There is a table of 40 tool length offset numbers, and one or more registers, each with a tool length offset modifier number. There is a tool length offset mode which can be set to one of three values: NONE, NORMAL, and MODIFIED. In the NONE mode, no tool length offsets are used. In the NORMAL mode, the tool length offset value in the position of the table with the same index as the tool currently in the spindle is used. In the MODIFIED mode, the value

used for the tool length offset is the modifier number in the currently selected modifier register added to the offset value for the tool currently in the spindle. There are currently no commands for setting the values in the tool length offset table or for setting the values of the modifier numbers in the modifier registers.

RS274/NGC Model - There is a large table of tool length offsets. The value stored in any position of the table can be selected as the current tool length offset, independent of which tool is in the spindle. Using one of the offsets is programmed using G43 with an H number (an integer giving the table position) or by including an H number with a tool change command. Using no tool length offset is programmed using G49.

The interpreter translates a G49 command from RS274/NGC into a USE_NO_TOOL_LENGTH_OFFSETS canonical machining function command.

When a G43 command is used, the interpreter reconciles the two models by behaving as follows.

1. If the H number programmed with the G43 is between 1 and 40, the USE_NORMAL_TOOL_LENGTH_OFFSETS() command is given and the tool_length_offset_register in the machine model is reset if changed. Also, if the previous tool length offset setting was not NORMAL, or if the new H number differs from the previous one, a new offset is being used. Since the interpreter does not know what the offset values are, the current z value becomes UNKNOWN.

2. If the H number is greater than 40, the USE_MODIFIED_TOOL_LENGTH_OFFSETS(H number) command is given. This indicates that the modifier value in the register given by the H number should be used. The tool_length_offset_register is reset if changed. As above, the current z value becomes UNKNOWN if any change has been made from the previous setting.

A.8 Order of Execution of a Block

In RS274/NGC, as in all dialects of RS274, a line (also known as a block) of code may specify several different things to do, such as moving from one place to another along a straight line or arc, changing the feed rate, starting the spindle turning, etc. The order of execution of items in a block is critical to safe and effective machine operation (it is a good idea to start the spindle before cutting, for example), but is not specified clearly in [Allen Bradley].

The output of the interpreter is not grouped in blocks. There is only one action per line (or function) and the lines (or functions) are executed in order. In the interpreter, output functions are called in the following order if they occur in the same block:

1. comment command.
2. set feed rate (F) command.
3. set spindle speed (S) command.
4. “M” commands greater than M2 in the order: first M6 (tool change), then one of M3, M4, or M5 (spindle on or off), then one of M7, M8, or M9 (coolant on or off).
5. explicit “g” commands - first those greater than G3 in increasing order, then one of G0, G1, G2, or G3, if present.
6. an implicit G0, G1, G2, or G3 command if there are no explicit G’s, and the current G mode is one of those four.
7. stopping commands (M0, M1, or M2).

Appendix B Canonical Machining Functions

This appendix describes the canonical machining functions used by the interpreter. Most descriptions given here are similar to the descriptions given in Appendix C of [Albus]. Several functions presented in [Albus] are not used by the interpreter, and a few functions used by the interpreter are not in [Albus].

The machining functions given here are listed in Table 1 of the main body of the report. In Table 1 they are given as functions in the C language. The descriptions in this appendix are language-independent.

B.1 Representation

B.1.1 SET_PROGRAM_ORIGIN *x, y, z*

Set the program origin at the point with absolute coordinates x, y, and z. Values of x, y, and z are real numbers. The units are whatever length units are being used at the time this command is given.

B.1.2 USE_ABSOLUTE_ORIGIN *(no parameters)*

Use the absolute machine origin.

B.1.3 USE_LENGTH_UNITS *units*

Use the specified *units* for length. Acceptable values of *units* are the strings “inches” and “millimeters”.

B.1.4 USE_PROGRAM_ORIGIN *(no parameters)*

Use the program origin whose absolute location is specified by a set_program_origin command.

B.2 Free Space Motion

B.2.1 SET_TRAVERSE_RATE *rate*

Set the traverse rate that will be used when the spindle traverses. It is expected that no cutting will occur while a traverse move is being made.

B.2.2 STRAIGHT_TRAVERSE *x, y, z*

Move in a straight line at traverse rate from the current point to the point given by the *x*, *y* and *z* parameters.

B.3 Machining Attributes

B.3.1 SELECT_PLANE *selected_plane*

Use the plane designated by *selected_plane* as the selected plane. Acceptable values of the string *selected_plane* are “XY”, “XZ”, and “YZ”.

B.3.2 SET_FEED_RATE *feed_rate*

Set the feed rate that will be used when the spindle is told to move at the currently set feed rate.

B.3.3 STOP_CUTTER_RADIUS_COMPENSATION *(no parameters)*

Do not apply cutter radius compensation when executing spindle translation commands.

B.4 Machining Functions

It is assumed in these activities that the spindle tip is always at some location called the "current location," and the controller always knows where that is. It is also assumed that there is always a "selected plane" which must be the xy-plane, the yz-plane, or the xz-plane of the machine.

B.4.1 ARC_FEED *first_axis_coordinate, second_axis_coordinate, rotation, axis_end_point*

Move in a helical arc from the current location at the existing feed rate. The axis of the helix is parallel to the x, y, or z axis, according to which one is perpendicular to the selected plane. The helical arc may degenerate to a circular arc if there is no motion parallel to the axis of the helix. If the selected plane is the xy-plane, *first_axis_coordinate* is the axis x-coordinate, *second_axis_coordinate* is the axis y coordinate, and *axis_end_point* is the z-coordinate of the end of the arc. If the selected plane is the yz-plane, *first_axis_coordinate* is the axis y-coordinate, *second_axis_coordinate* is the axis z-coordinate, and *axis_end_point* is the x-coordinate of the end of the arc. If the selected plane is the xz-plane, *first_axis_coordinate* is the axis x-coordinate, *second_axis_coordinate* is the axis z coordinate, and *axis_end_point* is the y-coordinate of the end of the arc. The *rotation* parameter represents the number of radians in the arc. *Rotation* is positive if the arc is traversed counterclockwise as viewed from the positive end of the coordinate axis perpendicular to the currently selected plane. The radius of the helix is determined by the distance from the current location to the axis of helix.

B.4.2 STRAIGHT_FEED *x, y, z, probe*

Move in a straight line at existing feed rate (or using the existing z-force) from the current point to the point given by the *x*, *y* and *z* parameters.

The *probe* parameter has a binary value which we will take to be either *off* or *on*. In the interpreter, only the *off* value is used. If *probe* is *on*, the feed motion will stop when the probe is tripped or when the endpoint is reached, whichever happens first. The probe must be in the spindle and turned on beforehand if *probe* is *on*. If *probe* is *on* and the probe is tripped during the motion, the fact that the probe was tripped and the position of the spindle at the time the probe was tripped will be recorded in the appropriate registers or variables.

If the probe is tripped, the end position of the spindle is not determined, except that it must lie on the straight line being followed, must lie between the trip point and the programmed end point, and must be within a certain distance of the trip point. This indeterminacy is necessary because the spindle cannot be stopped instantly and the controller cannot look ahead and slow it down as it nears the end, as it would normally do for other motions. It should be feasible to require the stopping point to be a fixed distance from the trip point, if that seems more useful. Alternatively, the spindle could double back and stop at the point at which the probe was tripped.

B.5 Spindle Functions

B.5.1 SET_SPINDLE_SPEED *speed*

Set the spindle speed that will be used when the spindle is turning. This is usually given in rpm and refers to the rate of spindle rotation. If the spindle is already turning and is at a different speed, change to the speed given with this command.

B.5.2 SPINDLE_RETRACT_TRAVERSE *(no parameters)*

Retract the spindle at traverse rate to the fully retracted position.

B.5.3 START_SPINDLE_CLOCKWISE *(no parameters)*

Turn the spindle clockwise at the currently set speed rate. If the spindle is already turning that way, this command has no effect.

B.5.4 START_SPINDLE_COUNTERCLOCKWISE *(no parameters)*

Turn the spindle counterclockwise at the currently set speed rate. If the spindle is already turning that way, this command has no effect.

B.5.5 STOP_SPINDLE_TURNING *(no parameters)*

Stop the spindle from turning. If the spindle is already stopped, this command may be given, but it will have no effect.

B.5.6 USE_NO_SPINDLE_FORCE *(no parameters)*

Do not use spindle force. Instead, use the feed rate to determine spindle motion.

B.6 Tool Functions

B.6.1 CHANGE_TOOL *slot_number*

It is assumed that each cutting tool in the machine is assigned to a slot (intended to correspond to a slot number in a tool carousel). This command results in the tool currently in the spindle (if any) being returned to its slot, and the tool from the slot designated by *slot_number* being inserted in the spindle. If there is no tool in the slot designated by *slot_number*, there will be no tool in the spindle after this command is executed. For the purposes of this command, the tool includes the tool holder.

B.6.2 USE_MODIFIED_TOOL_LENGTH_OFFSETS *register*

Use the tool length offsets given in the tool length offset registers as modified by the number in the given *register*. This command covers modifying tool length offsets to compensate for different fixtures or for moving the head of the machining center (if it has a movable head).

B.6.3 USE_NORMAL_TOOL_LENGTH_OFFSETS *(no parameters)*

Use the tool length offsets given in the tool length offset registers.

B.6.4 USE_NO_TOOL_LENGTH_OFFSETS *(no parameters)*

Do not use tool length offsets.

B.7 Miscellaneous Functions

B.7.1 COMMENT *comment_text*

This function has no physical effect. If commands are being printed or logged, the comment command is printed or logged, including the string which is the value of *comment_text*. This serves to allow formal comments at specific locations in programs or command files.

B.7.2 FLOOD_OFF *(no parameters)*

Turn flood coolant off.

B.7.3 FLOOD_ON *(no parameters)*

Turn flood coolant on.

B.7.4 MIST_OFF *(no parameters)*

Turn mist coolant on.

B.7.5 MIST_ON *(no parameters)*

Turn mist coolant on.

B.8 Program Functions

B.8.1 OPTIONAL_PROGRAM_STOP*(no parameters)*

If the machining center has an optional stop switch, and it is on when this command is read from a program, stop executing the program at this point, but be prepared to resume with the next line of the program. If the machining center does not have an optional stop switch, or commands are being executed with a stop after each one already (such as when the interpreter is being used with keyboard input), this command has no effect.

B.8.2 PROGRAM_END*(no parameters)*

If a program is being read, stop executing the program and be prepared to accept a new program or to be shut down.

B.8.3 PROGRAM_STOP*(no parameters)*

If this command is read from a program, stop executing the program at this point, but be prepared to resume with the next line of the program. If commands are being executed with a stop after each one already (such as when the interpreter is being used with keyboard input), this command has no effect.

Appendix C Transcript of a Session

This is a transcript of a session using the interpreter with keyboard input. The interpreter was built using the version of the canonical machining functions which prints function calls. Characters entered by the user are shown in **boldface**. A carriage return is indicated by <CR>.

1} **ngc_to_prim<CR>**
ready for automatic initialization
this will stop spindle turning and cause xyz motion
enter semicolon to continue, any other key to abort => **;<CR>**
initializing - please wait
STOP_SPINDLE_TURNING()
SET_SPINDLE_SPEED(0.000000)
SET_TRAVERSE_RATE(100.000000)
SET_FEED_RATE(0.000000)
FLOOD_OFF()
MIST_OFF()
USE_ABSOLUTE_ORIGIN()
SET_PROGRAM_ORIGIN(0.000000, 0.000000, 0.000000)
USE_LENGTH_UNITS(inches)
SELECT_PLANE(XY)
USE_NO_TOOL_LENGTH_OFFSETS()
STRAIGHT_TRAVERSE(20.000000, 10.000000, 0.000000)
USE_NORMAL_TOOL_LENGTH_OFFSETS()
initialization complete
READ => **g1 x3 y1 f20.0<CR>**
EXEC <-**;<CR>**
SET_FEED_RATE(20.000000)
STRAIGHT_FEED(3.000000, 1.000000, 0.000000, 0.000000)
READ => **g2 x[6-[4*3/2]] r 7.01 z0.5<CR>**
EXEC <-**;<CR>**
ARC_FEED(1.500000, 7.847635, -0.431295, 0.500000)
READ => **(that was a helical arc)<CR>**
EXEC <-**;<CR>**
COMMENT("that was a helical arc")
READ => **t2<CR>**
tool number on line with no tool change
command read error
READ => **m6 t2<CR>**
EXEC <-**;<CR>**
STOP_SPINDLE_TURNING()
SPINDLE_RETRACT_TRAVERSE()
CHANGE_TOOL(2)
READ => **m2<CR>**
EXEC <-**;<CR>**
PROGRAM_END()

Appendix D Error Messages

This is an alphabetical list of all error messages in the interpreter. Messages are in **boldface** type. Following each message is the name of the function or functions in which it is found in italics. The messages are generally self-explanatory. An ***a*** in message stands for a single character which varies according to what is being checked; similarly, ***n*** stands for an integer.

1. **arc radius too small**. . . . . *convert_arc_xy, convert_arc_yz, convert_arc_zx*
2. **attempt to divide by zero** . . . . . *read_integer_expression, read_real_expression*
3. **bad call to read_comment** . . . . . *read_comment*
4. **bad call to read_feed_rate** . . . . . *read_feed_rate*
5. **bad call to read_g** . . . . . *read_g*
6. **bad call to read_integer_expression** . . . . . *read_integer_expression*
7. **bad call to read_m**. . . . . *read_m*
8. **bad call to read_radius** . . . . . *read_radius*
9. **bad call to read_real_expression** . . . . . *read_real_expression*
10. **bad call to read_spindle_speed** . . . . . *read_spindle_speed*
11. **bad call to read_tool** . . . . . *read_tool*
12. **bad call to read_tool_length_offset** . . . . . *read_tool_length_offset*
13. **bad call to read_x** . . . . . *read_x*
14. **bad call to read_y** . . . . . *read_y*
15. **bad call to read_z** . . . . . *read_z*
16. **bad number format in read_integer** . . . . . *read_integer*
17. **bad number format in read_real** . . . . . *read_real*
18. **bad character '*a*' in read_one_item** . . . . . *read_one_item*
19. **bad character '*a*' in expression** . . . . . *read_real_expression*
20. **bad format line number** . . . . . *read_file_line, read_keyboard_line*
21. **bad character '*a*' in expression** . . . . . *read_integer_expression*
22. **cannot find input file *filename*** . . . . . *interpret_from_file*
23. **command execution error** . . . . . *interpret_from_file, interpret_from_keyboard*
24. **command read error**. . . . . *interpret_from_file, interpret_from_keyboard*
25. **duplicate g codes** . . . . . *g_codes_ok*
26. **duplicate m codes** . . . . . *m_codes_ok*
27. **g codes g*n* and g*n* from same modal group** . . . . . *g_codes_ok*
28. **g codes read incorrectly** . . . . . *g_codes_ok*
29. **h number on line with no tool change or offset** . . . . . *other_codes_ok*
30. **illegal nested comment found**. . . . . *close_and_downcase*
31. **line ends before expression**. . . . . *read_integer_expression*
32. **line ends before expression**. . . . . *read_real_expression*
33. **m codes read incorrectly**. . . . . *m_codes_ok*
34. **m codes m*n* and m*n* from same modal group** . . . . . *m_codes_ok*
35. **missing radius in convert_arc** . . . . . *convert_arc*
36. **more than three m codes** . . . . . *read_m*
37. **more than two g codes** . . . . . *read_g*
38. **multiple feed rate settings on one line**. . . . . *read_feed_rate*
39. **multiple r settings on one line** . . . . . *read_radius*
40. **multiple spindle speed settings on one line** . . . . . *read_spindle_speed*