NISTIR 5738

The NIST RS274KT Interpreter

Thomas R. Kramer
Research Associate

Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064
and
Intelligent Systems Division

and

Frederick Proctor
Intelligent Systems Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

NIST

NISTIR 5738

The NIST RS274KT Interpreter

Thomas R. Kramer
Research Associate

Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064
and
Intelligent Systems Division

and

Frederick Proctor
Intelligent Systems Division

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

October 1995

The NIST RS274KT Interpreter

Thomas R. Kramer
Frederick Proctor

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

NISTIR 5738
October 26, 1995

Disclaimer

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

The NIST "RS274KT interpreter" is a software system which reads numerical control code in the "KT" dialect of the RS274 numerical control language and produces calls to a set of canonical machining functions. The output of the interpreter can be used to drive a Kearney and Trecker 800 4-axis machining center. This report describes the RS274KT interpreter.

1.1 Background

1.1.1 Enhanced Machine Controller Project

The Intelligent Systems Division (ISD) and the Automated Production Technology Division of the National Institute of Standards and Technology (NIST) are carrying out an Enhanced Machine Controller (EMC) project. The primary objective of the project is to build a testbed for evaluating application programming interface standards for open-architecture machine controllers. A secondary objective is to retrofit a Kearney and Trecker (K&T) 800 4-axis machining center using the EMC open-architecture control system.

1.1.2 Numerical Control Programming Language RS274

RS274 is a programming language for numerically controlled (NC) machine tools, which has been used for many years. The most recent standard 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]. Most NC machine tools can be run using programs written in RS274. Implementations of the language differ from machine to machine, however, and a program that runs on one machine probably will not run on one from a different maker.

1.1.3 The RS274KT Language

The RS274KT language is described in the "Part Programming and Operating Manual" [K&T] for the K&T 800 machine. It dates from the late 1970's and differs only slightly from the standard RS274-D. The term "the manual" in this report always means that manual.

1.1.4 Previous Work at NIST

As part of ISD assistance to the program which developed the NGC architecture, ISD 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 was implemented in 1993 by the EMC project as a set of functions in the C programming language. This suite, known in the EMC project and in this report as the "canonical machining functions," has now been revised to be suitable for 4-axis machining.

Also in 1993, the authors developed a software system in the C language for reading machining commands in the RS274/NGC language (an extension of RS274 proposed by the NGC project) and outputting canonical machining functions. This was called "the RS274/NGC interpreter." A report, "The NIST RS274/NGC Interpreter, Version 1" [Kramer1] was published in 1994 describing that interpreter.

1.1.5 Current Work at NIST

In 1994, the EMC project, in collaboration with the General Motors Company (GM), undertook to retrofit the 4-axis K&T machine mentioned earlier with an EMC controller. This was the driving force in the revision of the canonical machining functions, and also led NIST to build the RS274KT interpreter. With the RS274KT interpreter, GM is able to continue to use existing NC programs on the K&T 800 and to continue to use their existing methods of generating NC programs for that machine.

Concurrently with the development of the RS274KT interpreter, a second version of the RS274/NGC interpreter was built which implements about twice as much of the RS274/NGC language as did the first version. A separate report [Kramer2] has been prepared documenting this new version of the RS274/NGC interpreter. The EMC controller can easily be reconfigured to use the RS274/NGC interpreter, should GM wish to use NC programs written in RS274/NGC.

The EMC project decided to have all software be written in a single language, C++, so the source code for both interpreters is now in C++. Almost all of the interpreter source code could be compiled by an ANSI C compiler, however. Little of the source code is in constructions used in C++ but not in C, such as data members and member functions.

1.2 Overview of RS274KT Code

The RS274KT language is very close to the standard definition of RS274-D.

1.2.1 Lines, Blocks, Commands, and Words

The RS274KT 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 typical line of code consists of an optional line number at the beginning followed by one or more "words," possibly interspersed with comments. In this report, a word consists of a letter followed by 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 RS274KT commands start with either G or M (for miscellaneous). The words for these commands are called "G codes" and "M codes." In addition to words, lines of code may include other combinations of characters. The legal combinations of characters for the RS274KT language are presented in Appendix E.

1.2.2 Commands and Machine Modes

In RS274KT, many G codes and M codes cause the machine to change from one mode to another, and the mode stays active until some other command changes it implicitly or explicitly [K&T, pages 4.3 - 4.5]. Such commands are called "modal". For example, the coolant commands are modal. If coolant is turned on, it stays on until it is explicitly turned off. The G codes for motion are also modal. If a G1 (straight move) 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).

"Non-modal" codes have effect only on the lines on which they occur. For example, M25 (bypass offsets) is non-modal.

1.2.3 Modal Groups

Modal commands are arranged in sets called "modal groups", and only one member of a modal group may be in force at any given time. In general, a modal group contains commands for which it is logically impossible for two members to be in effect at the same time — like measure in inches vs. measure in millimeters. A machine tool may be in many modes at the same time, with one mode from each modal group being in effect. The modal groups used in the interpreter are shown in Table 1.

For several modal groups, it is usual to provide 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.

The modal groups for G codes are:

group 1 = {G0, G1, G2, G3, G80, G81, G82, G83, G84, G85, G86, G87, G88, G89} - motion
group 2 = {G17, G18, G19} - plane selection
group 3 = {G90, G91} - distance mode
group 5 = {G93, G94} - spindle speed mode
group 6 = {G70, G71} - units
group 7 = {G40, G41, G42} - cutter diameter compensation
group 10 = {G98, G99} - axis offsets

The modal groups for M codes are:

group 2 = {M26, M27} - B-axis clamping
group 4 = {M0, M1, M2, M30, M60} - stopping
group 6 = {M6} - tool change
group 7 = {M3, M4, M5} - spindle turning
group 8 = {M7, M8, M9} - coolant
group 9 = {M48, M49} - feed and speed override switch bypass

Table 1. Modal Groups

1.3 Canonical Machining Functions

The EMC canonical machining functions called by the interpreter are given in Table 2 and are described in more detail in a separate report, "Canonical Functions for 4-Axis Machining" [Proctor]. That report includes additional canonical machining functions which are not used by the interpreter but may be in future versions. The names of the functions explain roughly what they do. The canonical machining commands are atomic commands. Each command produces a single action.

RS274 commands, on the other hand, include two types: those for which a single RS274 command corresponds exactly to a canonical command, and those for which a single RS274 command will be decomposed into several canonical commands (possibly dozens). Things like "move in a straight line" or "turn flood coolant on" are of the first type. Things like "turn all coolant off" or "run a peck drilling cycle" are of the second type.

The canonical commands were devised with three objectives in mind. First, all the functionality of the K&T 800 machine had to be covered by the commands; for any function the machine can perform, there has to be a way tell it to do that function. Second, it was desired that it be possible to use readily available commercial motion control boards from various vendors to carry out those canonical commands which call for motion, with roughly a one-to-one correspondence between a canonical motion command and a command a commercial board recognizes. Third, it must be possible to interpret RS274KT commands into canonical commands. The first and third objectives are closely related; if one of them can be reached, the other will probably also have been reached.

Two sets of definitions for the canonical 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 a command message to be generated. When this command message is executed, the machine's actuators are activated. 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_ORIGIN_OFFSETS (double x, double y, double z, double b) USE_LENGTH_UNITS (CANON_UNITS units)
Free Space Motion	STRAIGHT_TRAVERSE (double x, double y, double z, int b_turn, double b_position)
Machining Attributes	SELECT_PLANE (CANON_PLANE plane) SET_FEED_RATE (double rate) SET_FEED_REFERENCE (CANON_FEED_REFERENCE reference) START_SPEED_FEED_SYNCH() STOP_SPEED_FEED_SYNCH()
Machining Functions	ARC_FEED (double first_end, double second_end, double first_axis, double second_axis, int rotation, double axis_end_point, int b_turn, double b_position) STRAIGHT_FEED (double x, double y, double z, int b_turn, double b_position) DWELL (double seconds)
Spindle Functions	SET_SPINDLE_SPEED (double r) START_SPINDLE_CLOCKWISE () START_SPINDLE_COUNTERCLOCKWISE () STOP_SPINDLE_TURNING () ORIENT_SPINDLE (double orientation, CANON_DIRECTION direction)
Tool Functions	CHANGE_TOOL (int slot) SELECT_TOOL (int i) USE_TOOL_LENGTH_OFFSET (double offset)

Miscellaneous Functions	CLAMP_AXIS(CANON_AXIS axis)
	COMMENT (char * s)
	DISABLE_FEED_OVERRIDE()
	DISABLE_SPEED_OVERRIDE()
	ENABLE_FEED_OVERRIDE()
	ENABLE_SPEED_OVERRIDE()
	FLOOD_OFF ()
	FLOOD_ON ()
	MESSAGE (char * s)
	MIST_OFF ()
	MIST_ON ()
	PALLET_SHUTTLE()
	UNCLAMP_AXIS(CANON_AXIS axis)

Program Functions	OPTIONAL_PROGRAM_STOP () PROGRAM_END () PROGRAM_STOP ()

Table 2. Canonical Machining Functions Called By Interpreter
Function arguments are written in ANSI C style. All functions return nothing.

2 Overview of the Interpreter

2.1 Major Characteristics

2.1.1 Modes of Use

The interpreter runs integrated with the EMC control system or as a stand-alone system. Most of the software is the same in the two cases. Software details are given in Appendix A. The reader may find it helpful to look at Figure 1 in Appendix A at this point.

2.1.1.1 Integrated with EMC Control System

In the EMC control system, the interpreter is used both to interpret NC programs (from files) and to interpret individual commands entered using the manual data input (MDI) capability of the control system. When running an NC program, the control system tells the interpreter when to read another line of code from the program and when to execute the last line that was read. When using MDI input, the controller sends the interpreter a line of code it gets from the user interface with a single command that tells the interpreter to read the line and execute the line.

The interpreter does not control machine action directly. Rather, the interpreter calls canonical functions which generate messages which are passed back to the control system, and the control system decides what to do with the messages. In normal operation the top level of the control system decides whether each message should be sent to its motion control subordinate or to its discrete I/O subordinate and sends the message at an appropriate time.

If an error occurs, the user is sent an error message. If an NC program is being translated, execution of the program stops at the line where the error occurred, and it is not possible to restart the program from that line. To use a program which causes an interpreter error, the program must be edited to remove the error, and the program must be restarted at the beginning. The user must determine if the partially cut workpiece can be saved or whether it must be scrapped, and must consider the effect of re-running the portion of the program before the error occurred in making this decision. Of course the user has the option of running a revised program which deletes the code that ran successfully at the beginning of the original program.

2.1.1.2 Stand-alone

In stand-alone mode, the interpreter has two input modes: keyboard (interactive) mode and file (batch) mode. In the interactive mode, the user types lines of RS274KT code at the keyboard. In the batch mode, the interpreter reads lines of code from a file.

Only the set of canonical machining functions which prints text has been linked into the stand-alone interpreter, so the output is always text. The output is printed to the computer terminal by default, but may be redirected to a file. In general, an output file is useful only if input is taken from a file and errors do not occur during interpretation.

Error handling differs between the interactive mode and the batch mode. In interactive mode, the interpreter always continues after an error. In batch mode the user has an option when starting the interpreter of telling the interpreter to try to keep going after an error or having the interpreter stop interpreting if an error occurs (which is the default behavior).

The stand-alone mode is valuable because it allows a user to pre-test an NC program without having to run it on the machine controller itself. Any computer for which the stand-alone interpreter can be compiled can be used to pre-test NC programs. Pre-tests are conclusive tests because the interpreter runs exactly the same way in the stand-alone mode as it does integrated with the control system.

Section 4.1 explains how to use the various options available in stand-alone mode.

2.1.2 How it Runs

Once initialized, the interpreter runs in various ways (depending on which of the modes of use just described is being used) but in all of them, the basic operation that gets repeated is a two-step process:

1. Get a line of RS274KT code and read it into memory, building an internal representation of the meaning of the entire line.
2. Call one or more canonical machining functions which will do what the line says to do.

The interpreter maintains a model of the machine while it is interpreting and uses the model in determining what canonical machining functions to call and what their parameters should be. Initialization of the model is performed when the interpreter starts up.

2.1.3 Speed

2.1.3.1 Stand-alone 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 stand-alone interpreter wrote an output file in 4 seconds. A second file with 3600 small arcs lying on a helix took 9 seconds. A third file with 2000 straight line segments took 5 seconds. These three tests show a rate of 400 to 500 lines per second for the stand-alone interpreter on a Sun SPARC station 2 computer.

Running on a 486 PC using the Lynx operating system, the stand-alone interpreter handled the same three test files in 12 seconds, 23 seconds, and 13 seconds, about a third as fast as on the Sun SPARC. The output from the two computers was byte-by-byte identical for the first two input programs. For the third program there were about 100 numbers which differed in the last decimal place, apparently because of different conventions for rounding used by the two computers when the digit after the last decimal place to be kept is a 5.

Tests with other types of programs show similar results. Thus, it is clear that using the stand-alone system for pre-testing NC programs takes little time.

2.1.3.2 Interpreter Speed in the Integrated System

When the interpreter is being used integrated with the EMC control system, there are relatively few situations in which a modern PC would be challenged, since the tool path prescribed by a line found in a typical program will rarely take less than a tenth of a second to cut, while interpreting that line will rarely take more than a hundredth of a second. Thus, for most NC programs, the interpreter speed will not be a significant factor.

There is one known class of program for which interpreter speed may be a significant factor. This class is those programs which include many consecutive short lines or arcs (say 0.1 millimeter long each). This type of program is produced by many NC program post-processors for following complex contours approximately. At 1000 millimeters per minute feed rate (a realistic value), a 2000-line file should run in 12 seconds. Tests of this type of program on the integrated system have not yet been conducted, but the results just mentioned for the stand-alone interpreter running on a PC indicate that even without all the other tasks the integrated system must perform, this speed challenges the interpreter. The interpreter has not been optimized for speed. If it is decided more speed is required during execution, that may be accomplished by various types of pre-processing which are not difficult to implement.

2.2 Start-up

When the interpreter starts up, before accepting any input, it sets up a model that includes data about itself, data about the setup of the machine to be controlled, and data about the tools which are in the tool carousel of the machine. The interpreter's data about itself (such as whether tool radius compensation is on) has default values. The data about the machine and the tools is obtained in two different ways depending upon whether the interpreter is integrated or stand-alone.

In the integrated mode, machine setup data for the model is obtained by the interpreter from the (real) control system database. This data (such as axis positions) has been read from the sensors on the machine and represents the actual state of the machine. Also in this mode, the data about tools is obtained from the information that has been input by an operator.

In the stand-alone interpreter, machine setup data and tool data for the model is obtained (using exactly the same queries) from a stub version of the database of the control system which has default values for the data the interpreter needs. The default data does not necessarily represent the actual state of the machine or the tools in the carousel.

In the stand-alone interpreter, all the machine setup data and the tool data may be replaced by data from files designated by the user. In this way the stand-alone interpreter can be given data representing different conditions of the machine and tools; the data in the files may be actual or hypothetical. When the interpreter starts up in the stand-alone mode, the user is prompted to provide the name of a setup file and the name of a tool file. In both cases, if a name is provided, the data from the file is used and if a name is not provided, the data already loaded is used. The formats for setup files and tool files are described in Appendix F and Appendix G, respectively, with examples of both types of file.

Default setup data is shown in Table 3. Where G codes and M codes are involved, the settings are as provided at start-up in [K&T, page 4.3], except that G0 (rapid traverse) is not active at start-up in the interpreter.

Default tool data is shown in Table 4. The default tool table is not intended to be useful for preparing or testing any real NC programs. Data has been placed in the default tool table so the interpreter can be used without having to prepare a tool table. The default tool table has the tool length offset and the tool diameter as zero for all slots except slots 1 and 2.

The values in the tool table are used as though they are in the units (inches or millimeters) currently in use; the tool table values are not adjusted automatically if units are changed. This is not explicit in the manual but corresponds to the way the Kearney and Trecker machine works with its original controller.

Item	Setting	RS274KT code	Internal/External
axis_offset_b	0.0	G99	I
axis_offset_x	0.0	G99	I
axis_offset_y	0.0	G99	I
axis_offset_z	0.0	G99	I
block delete switch	off	console switch	E
b-axis clamp	on		E
b-axis automatic clamping	on	M26	I
current_b	0.0		E
current_x	0.0		E
current_y	0.0		E
current_z	0.0		E
cutter radius compensation	off	G40	I
distance mode	absolute	G90	I
feed mode	units per minute	G94	I
feed override	enabled	M48	I
feed rate	15		I
flood coolant	off	M9	E
length units	millimeters	G71	I
mist coolant	off	G9	E
motion mode	80	G80	I
plane for arcs	XY plane	G17	E
slot in use	1		E
slot selected	1		E
slot for length offset	1		I
slot for radius comp	1		I
speed_feed_mode	independent		I
speed override	enabled	M48	I
spindle speed	0.0		E
spindle turning	not turning	M5	E
tool length offset	0.0		I
traverse rate	100		E

Table 3. Default Setup Data for the Interpreter

Slot	ID	Length	Diameter
1	1	2.0	1.0
2	2	1.0	0.2

Table 4. Default Tool Data for the Interpreter
All other slots have the id, length, and diameter set to zero.

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 a message describing the error. Behavior after printing an error message depends upon the mode in which the interpreter is being used and was described in Section 2.1.1. 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 only if it reads a line with the one word "quit". Most variations of "quit" are valid, e.g., "Q uI t". In any other mode, the interpreter exits when it reads a line with an M2 (program exit) or M30 (program exit with tape rewind and pallet shuttle) command on it. The rest of the line is executed before the interpreter exits. If there are more lines in the file following the line which causes a program exit, they are ignored.

As stated earlier, in some modes which use file input, the interpreter will also exit if a line is read that causes an error. It is an error for the file to come to an end without an M2 or M30 command.

3 Building a Stand-Alone Executable

On a SUN SPARCstation 2, an executable file for the stand-alone interpreter may be built from source code in about 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, six source code files must be placed in the same directory along with the Makefile shown in Table 5 below. The source code files are:

canon.cc
canon.hh
rs274kt.cc
rs274kt.hh
driver.cc
nml_emc.hh

The first two files are the function definitions and header file for the canonical machining functions. The version of canon.cc which prints function calls is used with the stand-alone interpreter. The second two are the function definitions and header file for the interpreter kernel and interface functions. The last two are the function definitions and header file needed for the stand-alone interpreter that are not needed in the integrated interpreter.

An executable file named “rs274kt” is built in the same directory by giving the command:
make rs274kt.

In the Makefile, we are using the Gnu C++ compiler, “g++.” Any other C++ compiler may be substituted for g++.

```
canon.o: canon.cc canon.hh nml_emc.hh rs274kt.hh
        g++ -c -v -g -O canon.cc
rs274kt.o: rs274kt.cc canon.hh nml_emc.hh rs274kt.hh
        g++ -c -v -g -O rs274kt.cc
driver.o: driver.cc canon.hh nml_emc.hh rs274kt.hh
        g++ -c -v -g -O driver.cc

rs274kt: rs274kt.o canon.o driver.o
        g++ -v -o rs274kt rs274kt.o canon.o driver.o -lm
```

Table 5. Makefile for Stand-Alone Interpreter

4 Using the Stand-Alone Interpreter

4.1 Invoking the Interpreter

As mentioned earlier, the stand-alone interpreter may be used with either keyboard input or file input. This section tells how to do that. The user must press the “return” key after each response; the return key presses are not printed here.

This section describes several input files. If any input file named by the user cannot be opened, the interpreter prints a message to that effect and quits.

This section refers to redirecting output. The methods described here work with both the SunOS and LynxOS and may be expected to work on other Unix-like systems. Other forms of redirection are possible.

4.1.1 Invocation with Keyboard Input

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

rs274kt

4.1.2 Invocation with NC File Input

4.1.2.1 Invocation to Stop After an Error

To use NC file input and stop if an error is encountered, invoke the interpreter with a command of the form:

rs274kt ***input_filename***

where input_filename is the name of the NC input file. With this invocation, normal printed output from the interpreter (everything but error messages) appears on stdout, which is normally the terminal on which the command was invoked. Printed output may be usefully redirected to an output file by giving a command of the form:

rs274kt ***input_filename* > *output_filename***

where output_filename is the name the output file should have. The interpreter will create this file if it does not exist; if it does exist, it will be overwritten.

4.1.2.2 Invocation to Continue After an Error

To use file input and attempt to continue if an error is encountered, invoke the interpreter with a command of the form:

rs274kt ***input_filename*** **continue**

As above, normal printed output from the interpreter (everything but error messages) appears on stdout. Error messages are printed to stderr, which is also normally the terminal from which the interpreter was invoked. Printed output (excluding error messages) may be redirected to an output file by giving a command of the form:

rs274kt ***input_filename*** **continue >** ***output_filename***

If there are errors during interpretation, any input line which causes an error will not be interpreted, and the output file may be incorrect on any line after the last line which was output before the first error occurred.

4.2 Tool and Setup Files

As soon as the interpreter is invoked by any of the methods just described, it prompts the user to enter the name of a tool file. The user may enter a tool file name, as follows:

`name of tool file =>` *tool_file_name*

or the user may press only the return key, and the interpreter will use the default tool data.

The interpreter next prompts the user for the name of a setup file. The user may enter a setup file name as follows:

`name of setup file =>` *setup_file_name*

or the user may press only the return key, and the interpreter will use the default setup data.

The format of setup files and tool files is described in Appendix F and Appendix G, respectively.

4.3 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, after start-up, is a read-execute cycle with four steps:

1. The interpreter prints the prompt **`READ =>`**
2. The user enters a line of RS274KT code at the keyboard and hits the carriage return button.
3. The interpreter reads the line and prints the prompt **`EXEC <-`**
4. The user enters a semicolon and hits the carriage return button, and the line is interpreted.

Steps 3 and 4 have been included in the interface to give the user a chance to check the line just typed. If anything but a semicolon is entered in step 4, the line is not interpreted. This will protect the user who accidentally hits the return button during step 2.

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 INPUT

5.1 Overview

In general, allowable inputs are as described in the manual. [EIA], the standard for RS274-D is used where the manual is silent, but [EIA] has something to say.

5.1.1 White Space

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

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

5.1.2 Case Sensitivity

The manual and [EIA] do not explicitly discuss character case. 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.

5.2 Input Lines

To make the specification of an allowable line of code precise, we have defined it in a production language (Wirth Syntax Notation) in Appendix E. The description here is intended to be consistent with the appendix. In order that the definition in the appendix not be unwieldy, many constraints imposed by the interpreter are omitted from that appendix. The list of error messages in Appendix D indicates all of the additional constraints.

5.2.1 Format of a Line

A word is defined to be a letter followed by a number. A permissible line of input consists of the following, in order, with the restriction that there is a maximum (currently 256) to the number of characters allowed on a line.

1. an optional block delete character, which is a slash / .
2. an optional line number.
3. any number of segments, where a segment is a word or a comment.
4. an end of line character.

5.2.2 Number

The manual is not clear regarding what a valid number is. In some places, such as [K&T, page 3.7], it is stated that numbers must have decimal points and have different required formats depending on whether inches or millimeters are being used. In many of the examples in the manual there are no decimal points at all. Programs being used by GM use decimal points. [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 regarding numbers. In these 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.
- Numbers may have any number of digits, subject to the limitation on line length.
- A non-zero number with no sign as the first character is assumed to be positive.

Notice that initial (before the decimal point and the first non-zero digit) and trailing (after the decimal point and the last non-zero digit) zeros are allowed but not required. A number written with initial or trailing zeros will have the same value when it is read as if the extra zeros were not there.

Every number used in the RS274KT language is either an unsigned integer or a decimal number. An unsigned integer may not have a plus or minus sign in front of it or a decimal point in it; a decimal number may.

Different types of numbers may be required in different types of words. For example, in G words and M words, only unsigned integers between 0 and 99 are allowed.

5.2.3 Line Number

A line number is the letter N followed by an unsigned integer between 0 and 99999. This is a slight variation from [K&T, page 4.1], which allows numbers from 1 to 999999, but does not allow selecting a line number greater than 99999 while editing on the machine console.

5.3 Word Repeats

[K&T, page 4.4 and elsewhere] specifies that up to four G words and four M words may appear on a line, but is not explicit about words beginning with other letters. The interpreter uses the following rules:

- A line may have zero to four G words. Two G words from the same modal group may not appear on the same line.
- A line may have zero to four 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.

5.4 Word order

The manual does not specify word order explicitly or implicitly. The interpreter allows words starting with any letter except N (which denotes a line number and must be first) to occur in any order. Execution of the line will be the same regardless of the order.

5.5 Measurement Units

5.5.1 Linear units

[K&T page 3.7 and elsewhere] specifies that either "inch" or "metric" units may be used and makes it clear that the metric length units are millimeters. Linear units are selected by programming G70 for inch or G71 for metric. One or the other is automatically to be chosen when the system starts up; which one is a matter of choice [K&T, page 4.3]. The stand-alone interpreter starts up using millimeters, as shown in Table 3.

5.5.2 Angular units

[K&T, page 3.7 and elsewhere] specifies using degrees for measuring angles, so the interpreter assumes angular dimensions in the input are given in degrees.

5.6 Messages and Comments

[K&T, page 4.2] prescribes using parentheses to give messages, though the section is unclear. It does not specify whether nested pairs of parentheses are allowed, whether things other than messages are allowed inside parentheses, or whether a line which contains a left parenthesis can end without a right parenthesis somewhere after the left parenthesis. In the interpreter, a left parenthesis can occur anywhere on a line where the start of a word is allowed. There must be a closing right parenthesis on the line. A second left parenthesis may not occur before the closing right parenthesis (so nested parentheses are illegal).

In the interpreter, it is allowed to have more than one set of parentheses on a line (like) (this), but if this is done, only the last one will be processed as described below. All the others will be read and their format will be checked, but they will be ignored thereafter.

5.6.1 Messages

If "MSG," appears after a left parenthesis before any other printing characters, the rest of the characters before the right parenthesis are considered to be a message, and the "message" canonical function is called to deliver the message to the operator. Variants of "MSG," which include white space and lower case characters are allowed.

5.6.2 Comments

If the characters inside parentheses are not a message, as just described, then everything inside the parentheses is treated as a comment, and the "comment" canonical function is called. Comments do not cause the machine to do anything.

5.7 Programs

By "program" we mean a sequence of lines of RS274KT NC code (at least two and perhaps as many as several thousand) that are intended to be executed one after another. The sequence of lines is normally kept in a file.

The stand-alone interpreter has no concept of a program when it is running with keyboard input, it only understands lines. When it runs with input from a file, it expects the file to be an NC program. The interpreter checks that the first line of the file contains nothing but a percent sign (%), and it exits when a line with M2 or M30 is executed, since they mean end of program.

In the integrated interpreter, programs are recognized and handled by the user interface of the EMC control system.

The use of "%" as the only character on the first line of a program does not appear to be required in the manual, but is standard practice at GM, so it is a required feature in the interpreter.

5.8 Control Panel Switches

The EMC controller is sensitive to all switches on the control panel. The interpreter needs to know the setting of only one switch, block delete. The other switches are handled by the EMC controller without the interpreter knowing what the settings are.

5.8.1 Block Delete Switch

If the block delete switch is on, the interpreter skips lines which start with a slash (the block delete character). Internally, the interpreter reads the line but does not try to figure out what it means. If the switch is off, lines starting with a slash are processed as though the slash was not there.

When the interpreter is used as part of the EMC control system, information about the block delete switch setting is passed to the interpreter during the initialization of executing a program. The interpreter does not check again, so changing the setting of that switch during the execution of a program will not change the way the program is executed.

When the interpreter is used stand-alone, a block delete switch setting may be included in the setup file. The default setting of the block delete switch is off, as shown in Table 3. The default is used when a setting is not specified in a setup file.

5.8.2 Other Switches

The speed or feed override switches on the control panel let the operator specify that the actual feed rate or spindle speed used in machining should be some percentage of the programmed rate. The EMC control system reacts to the setting of the speed or feed override switches on the control panel, when those switches are enabled. It does this, however, without the interpreter ever knowing their settings. As mentioned elsewhere, the interpreter will interpret M48 and M49 commands which enable or disable the switches, but it does not need to know what their settings are to do that. In Table 3 there is no representation of the settings of these switches.

The optional program stop switch on the machine console works as follows. If this switch is on **and** an input line contains an M1 code, program execution is supposed to stop until the cycle start button is pushed. The interpreter interprets an M1 on an input line into an OPTIONAL_PROGRAM_STOP canonical command in the output, as described elsewhere in this report. The interpreter itself has no knowledge of the setting of the optional program stop switch. In Table 3 there is no representation of the setting of this switch. It is up to the rest of the EMC control system to check the optional stop switch when the optional_program_stop canonical command is executed and either stop or not.

6 Capabilities of the RS274KT Interpreter

The interpreter implements almost all of the RS274KT language. The current version includes the following capabilities. Any capability not explicitly included is excluded. Trying to use any excluded capability in an NC program will cause an error in the interpreter.

6.1 Words Recognized

The interpreter recognizes words beginning with the letters shown in Table 6. The meanings of the letters are as given in [K&T, page 2.8 and elsewhere].

Letter	Meaning
B	b-axis of machine
D	tool radius compensation number
E	incremental feed distance with G83 or G87
F	feedrate or dwell time
G	general function — see Section 6.2
I	x-axis center (or offset) for arcs with axis parallel to y-axis or z-axis pitch for a helical arc with axis parallel to x-axis
J	y-axis center (or offset) for arcs with axis parallel to x-axis or x-axis pitch for a helical arc with axis parallel to y-axis
K	z-axis center (or offset) for arcs with axis parallel to x-axis or x-axis pitch for a helical arc with axis parallel to z-axis
M	miscellaneous function — see Section 6.2
N	line number
P	tool length compensation slot number
R	canned cycle plane
S	spindle speed
T	tool selection
X	x-axis of machine
Y	y-axis of machine
Z	z-axis of machine

Table 6. Letters Recognized by the Interpreter

6.2 Input G Codes and M Codes

This section describes the specific G codes and M codes which have been implemented in the interpreter. Where the implementation differs from what is described in the manual, the differences are described. G and M codes given in the manual which have not been implemented are listed.

Decisions on what to implement and what not to implement were based on:

- the capabilities of the K&T 800 machine at GM
- examination of 24 GM programs for the machine provided to NIST by GM
- discussions with NC programmers at GM

The interpreter is intended to implement all the G and M codes which are currently used in K&T 800 programs at GM. The effect of any program prepared using the current GM methods and conventions should be the same when the program is run using the EMC controller (which uses the interpreter) as if the original K&T controller were used. The interpreter can interpret completely all 24 programs provided by GM (except where there are errors in the programs).

6.2.1 G Codes Implemented

The G codes shown in Table 7 have been implemented. Unless noted otherwise in Section 6.2.2, each G code is implemented as described in the manual.

G Code	Meaning
G0	rapid positioning
G1	linear interpolation
G2	circular/helical interpolation (clockwise)
G3	circular/helical interpolation (counterclockwise)
G4	dwell
G17	xy plane selection
G18	xz plane selection
G19	yz plane selection
G40	cancel cutter diameter compensation
G41	start cutter diameter compensation left
G42	start cutter diameter compensation right
G70	inch system selection
G71	millimeter system selection
G80	cancel motion mode (including any canned cycle)
G81	drilling canned cycle
G82	spotfacing canned cycle
G83	chip breaking canned cycle
G84	tapping canned cycle
G85	reaming canned cycle
G86	boring canned cycle
G87	deep hole drilling canned cycle
G88	boring (keylock) canned cycle
G89	feed in, dwell, feed out canned cycle
G90	absolute distance mode
G91	incremental distance mode
G93	inverse time feed mode
G94	feed per minute mode
G98	insert axis offset
G99	cancel axis offset

Table 7. G Codes Implemented in the Interpreter

6.2.2 Differences from the Manual Regarding G Codes

This section describes where the implementation of G codes differs from the manual. If a difference simply adds capability, so that it has no effect on the operation of existing programs, it is marked “EXTENSION”.

6.2.2.1 G0 and G1

G0 and G1 both mean to move in a straight line from the current location to the end point. In most dialects of RS-274, G0 is used for rapid traverse (not cutting) and G1 is used for cutting. In the manual, the meanings of G0 and G1 and differ from what is usual. The only difference between G0 and G1 in the manual is that in G0 mode, the K&T controller does not start dealing with the next motion (if there is one) "until the axes have positioned to within 0.001 inch of the end point" [K&T, page 3.1] whereas in G1 mode "the control does not wait" [K&T, page 3.2]. With both G0 and G1, the manual uses the convention that if the feed rate is set to zero, this means to move at rapid traverse rate [K&T, page 3.18].

The way GM programs use G0 and G1 is as close to the usual way as feasible under the rules in the manual. That is, rapid traverse is done only in G0 mode (with the feedrate set to zero), and straight line cutting is done only using G1, with the feed rate set to some positive value.

The interpreter differs from the manual regarding G0 and G1 in the following ways.

a. G0 runs only with the feed rate set to zero, and does a rapid traverse in that case. If G0 is programmed without the feed rate set to zero, an error occurs.
b. G0 does not cause the system to wait until the axes have positioned to within 0.001 inch of the end point before dealing with the next motion.
c. G1 does not cause rapid traverse when the feed rate is zero. If G1 is programmed when the feed rate is zero, an error occurs.
d. G0 and G1 are in the same modal group as all other motion. [K&T, page 4.4] puts G1 in same modal group as G0 only, and there is a separate modal group for other motion.

6.2.2.2 G2 and G3 Circular or Helical Motion

a. EXTENSION: A b-axis value is legal with G2 and G3. It is illegal in [K&T, page 3.29].
b. EXTENSION: A G2 or G3 move including z-axis motion is legal with cutter length compensation on. It is illegal in [K&T, page 3.29].
c. EXTENSION: Pitch words (k-word when XY-plane selected, i-word when YZ-plane selected, j-word when XZ-plane selected) are optional for helical arcs of 360 degrees or less. Pitch words are required for all helical arcs in [K&T, page 3.58]. If a pitch word is used, the length of the given axial move must be the same as the length which may be calculated from knowing the pitch and the final radial location, for some number of turns around the helix.
d. G2 and G3 are in the same modal group as all other motion. [K&T, page 4.4] puts G2 and G3 in a modal group by themselves.

6.2.2.3 G40 Cutter Diameter Compensation Cancel

a. EXTENSION: The ramp-off can be done along an arc. This is illegal in [K&T, page 4.24].
b. EXTENSION: The G40 does not need to appear on a line which includes an x or y word. It does in [K&T, page 4.24].

6.2.2.4 G41 and G42 Cutter Diameter Compensation

The kind of corner rounding done by the implementation (excluding ramp-on and ramp-off) is the same as what is shown in many diagrams in the manual [K&T, pages 4.25 - 4.61]. The manual does not describe the algorithm used by the current K&T controller, however, and it is possible there are differences.

a. The ramp-on [K&T, page 4.21] and ramp-off [K&T, page 4.22] paths might differ from those described in the manual [K&T, page 4.21]. It is not possible to tell from the manual what the actual ramp-on and ramp-off algorithms are. In the manual, it looks like the ramp-on path segment may make a sharp corner with the following path segment in some cases where the implementation will make the transition smooth (no change in direction at the junction).

b. A D value may not be specified on a line which does not have a G41 or a G42. This is allowed in the manual [K&T, page 4.24].

c. EXTENSION: The ramp-on can be done along an arc. This is illegal in the manual [K&T, page 4.24].

d. EXTENSION: The G41 or G42 does not need to occur on a line with an X or Y value. It does in the manual [K&T, page 4.24].

e. It is not possible to switch to a different D value while compensation is on; an error message will be given. It is possible in the manual [K&T, page 4.25].

f. It is not possible to switch directly from compensation left to compensation right; an error message will be given. It is possible in the manual [K&T, page 4.24].

g. It is not possible to command the tool to fit into a corner into which it will not fit; an error message will be given. It is possible but strongly discouraged in the manual [K&T, pages 4.26 - 4.30].

h. It is illegal to use G0 with compensation on. It is legal in the manual [K&T, page 4.24].

6.2.2.5 G88 Boring Cycle (Keylock)

a. Although G88 is included in the implementation, the keylock feature is not currently used. The spindle is actively oriented, instead.

6.2.2.6 G90 and G91, Absolute vs. Incremental length

a. EXTENSION: In the implementation, it is legal to switch from G90 to G91 or vice versa while tool length compensation is on. This is illegal in the manual [K&T, page 4.19].

6.2.2.7 G93 Inverse (Time) Feed Rate Coding

a. Inverse time feed rate coding cannot be used with cutter diameter compensation. This is an unlikely combination, anyway.

6.2.2.8 G98 and G99 Offset Control

a. Offsets cannot be changed while cutter radius compensation is active or while a canned cycle is active.

6.2.3 Input M Codes Implemented

The following M codes are implemented as described in the manual, with two exceptions:

a. Tool changes [K&T, pages 4-9, 4-10] are performed as described in Appendix B.3.

b. In the case of M25, both G0 and G91 must be in effect when M25 is used.

M Code	Meaning
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
M25	bypass offsets
M26	enable automatic b-axis clamping
M27	disable automatic b-axis clamping
M30	program end, pallet shuttle, and reset
M48	enable speed and feed overrides
M49	disable speed and feed overrides
M60	pallet shuttle and program stop

Table 8. M Codes Implemented in the Interpreter

6.3 Codes Not Implemented in the Interpreter

The following G codes listed in the manual are not implemented: G38, G50, G51.

The following M codes listed in the manual are not implemented: M19, M23, M24, M40, M57, M58, M59.

The H (horsepower) code is not implemented.

7 Limitations of the Interpreter

This section describes significant limitations of the interpreter.

The codes listed in the manual which are not implemented are listed just above in Section 6.3.

The interpreter does not save input lines. This means that RS274KT 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.

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
[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
[K&T]	Kearney and Trecker Co.; *Part Programming and Operating Manual, KT/CNC Control, Type C*; Pub 687D; Kearney and Trecker Corp.; 1980
[Kramer1]	Kramer, Thomas R.; Proctor, Frederick M.; Michaloski, John L.; *The NIST RS274/NGC Interpreter, Version 1*; NISTIR 5416; National Institute of Standards and Technology, Gaithersburg, MD; April 1994
[Kramer2]	Kramer, Thomas R.; Proctor, Frederick M.; *The NIST RS274/NGC Interpreter, Version 2*; NISTIR 5739; National Institute of Standards and Technology, Gaithersburg, MD; October 1995
[Proctor]	Proctor, Frederick M.; Kramer, Thomas R.;Michaloski, John L.; *Canonical Functions for 4-Axis Machining*; NISTIR in draft; National Institute of Standards and Technology, Gaithersburg, MD; May 1995

Appendix A Software

This appendix describes the software for the interpreter. The appendix is intended for users and programmers who might want to modify the software or simply understand it.

A.1 Software Modules and Function Call Hierarchies

The interpreter is written in C++. The program files are:

1. rs274kt.cc and its header file, rs274kt.hh — 7177 lines
2. canon.cc and its header file, canon.hh — 788 lines
3. driver.cc and its header file, nml_emc.hh — 1048 lines

Two methods of using the interpreter, stand-alone and integrated with the rest of EMC, are provided. The use of program files differs between the two methods. Some code is common to both, and some code differs.

The organization of the software by module and file is shown in Figure 1. In the figure, arrows show the direction of function calls. The stand-alone interpreter uses the files shown in the left and middle columns. The integrated interpreter uses the files shown in the right and middle columns. The stand-alone files are arranged so that they mimic the arrangement of the integrated interpreter. The mimicry is so that running the stand-alone provides as good a test as possible of running integrated. If there were no integrated interpreter, the EMC emulation functions and data shown in the left column would not be needed.

In terms of lines of code, excluding the EMC files shown in the right column, about two thirds of the code is in the kernel functions.

The rs274kt.cc file has two parts, kernel functions and interface functions. In the file itself the interface functions are given after the kernel functions. As shown in Figure 1, kernel functions are called only by interface functions, while interface functions are called by and call back to either EMC functions (if integrated) or driver functions (in the stand-alone). Both kernel functions and interface functions call canonical functions.

The driver.cc file also has two parts. In the file itself, the driver functions are given after the EMC emulation functions. The driver functions provide the user interface, and the EMC emulation functions provide an EMC-like environment for modeling data about the machine tool and controller settings.

Different versions of the canonical functions are used in the stand-alone and integrated interpreters. The canonical functions used with the stand-alone simply print themselves, while those used integrated send messages describing themselves. The same header file is used with both versions of the functions.

The source code is assembled automatically from pre-source code. The pre-source code is used for both the RS274KT interpreter and the RS274/NGC interpreter. In the pre-source code, each function is kept in a separate file, and there are three separate directories.

1. software common to both interpreters, which includes three subdirectories with all of the driver, all of the interface functions, and about a third of the kernel. There are many *#if defined* sections within individual functions in this directory for differences between the two interpreters.

2. header files, utilities and functions for the RS274KT interpreter only.
3. header files, utilities and functions for the RS274/NGC interpreter only.

When the pre-source code is assembled into source code, the *#if defined* sections are preprocessed so that only the appropriate sections appear in the source code. Figure 1 shows the source code setup, not the pre-source.

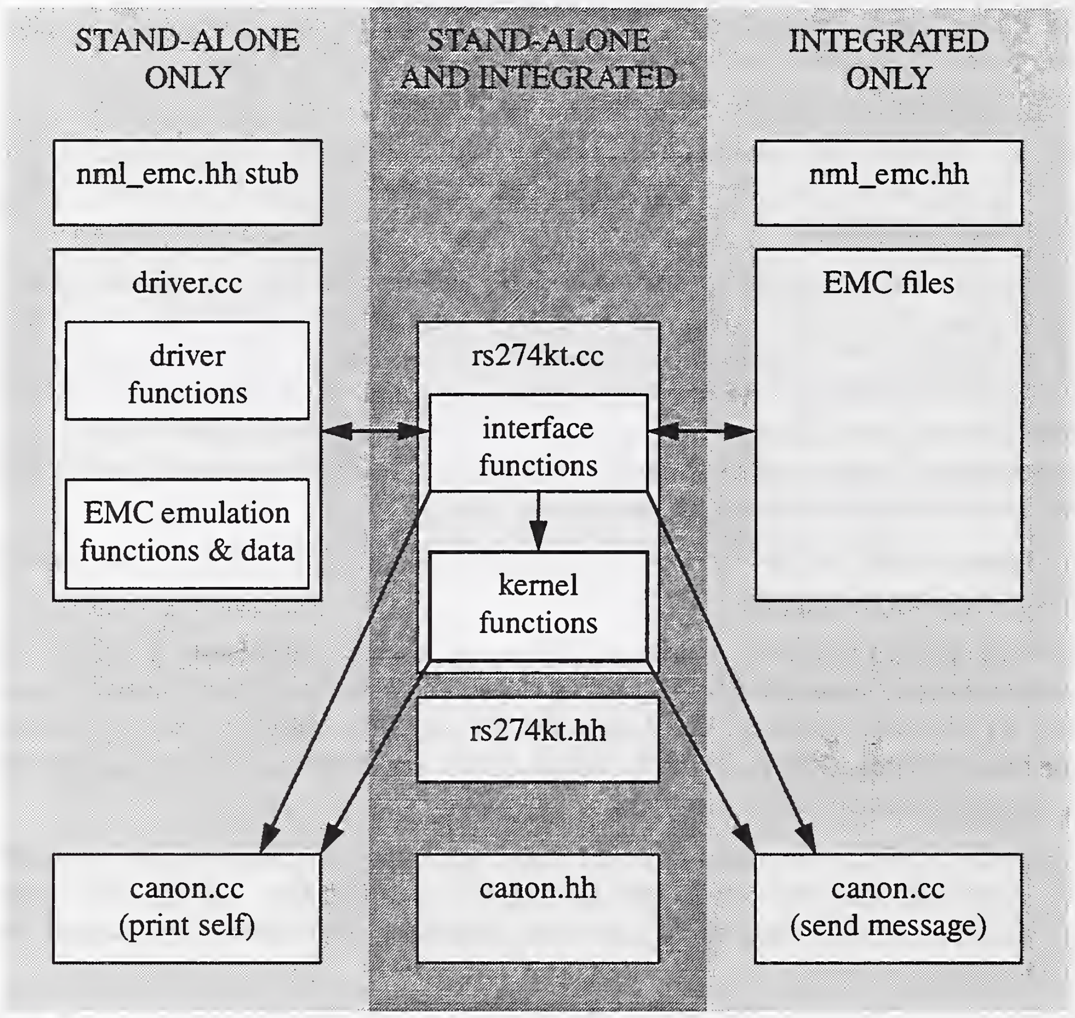

Figure 1. Interpreter Software

Function call hierarchies are shown in the following four figures. Figure 2 and Figure 3 show kernel function calls. All kernel functions appear on one or both of the two figures. Figure 4 shows the function calls from the interface functions. Figure 5 shows the hierarchy of function calls from the driver.

```
read_text —— close_and_downcase

read_line ___ check_items ___ check_g_codes
          |                |— check_m_codes
          |                └— check_other_codes
          |— init_block
          |— read_items ——┬— read_line_number —— read_integer_unsigned
          |               └— read_one_item ______ read_b ________________ read_real
          |                                    |— read_comment
          |                                    |— read_d ________________ read_integer_unsigned
          |                                    |— read_delta ____________ read_real
          |                                    |— read_f ________________ read_real
          |                                    |— read_g ________________ read_integer_unsigned
          |                                    |— read_i ________________ read_real
          |                                    |— read_j ________________ read_real
          |                                    |— read_k ________________ read_real
          |                                    |— read_m ________________ read_integer_unsigned
          |                                    |— read_r ________________ read_real
          |                                    |— read_spindle_speed ____ read_real
          |                                    |— read_tool_id __________ read_integer_unsigned
          |                                    |— read_tool_length_offset — read_integer_unsigned
          |                                    |— read_x ________________ read_real
          |                                    |— read_y ________________ read_real
          |                                    └— read_z ________________ read_real
          └— utility_enhance_block
                          └— utility_in_range
```

Figure 2. Interpreter Kernel Function Call Hierarchy (all but execute_block)

This shows the hierarchy of function calls from interpreter kernel functions read_text and read_line defined in rs274kt.cc to other kernel functions (excluding utility_error_number and nml_log_error - see text). Canonical functions calls are not shown. The hierarchy of calls from execute_block is shown in Figure 3.

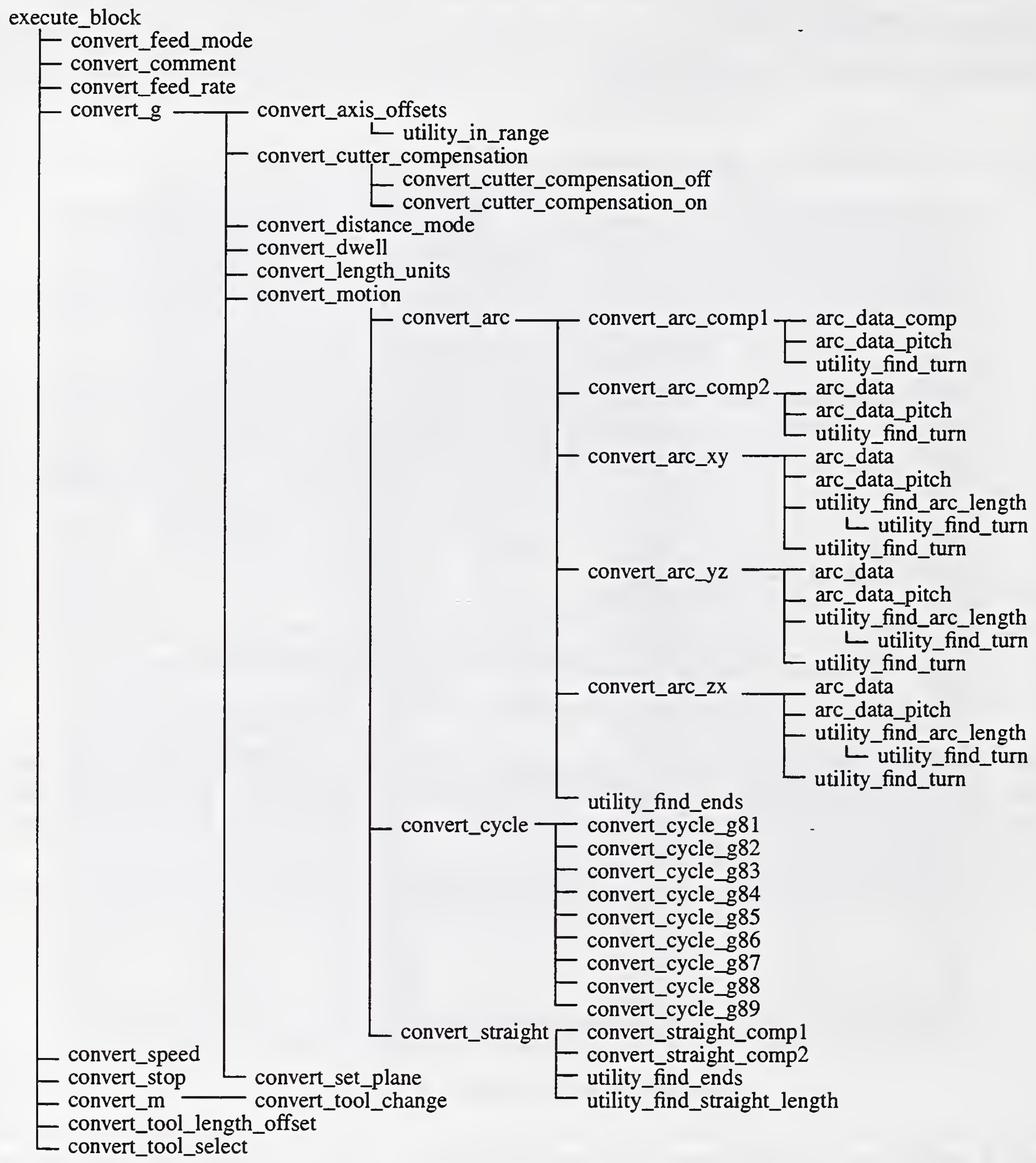

Figure 3. Interpreter Kernel Function Call Hierarchy (execute_block only)

This shows function calls from execute_block, defined in rs274kt.cc, to other kernel functions (excluding utility_error_number and nml_log_error - see text). Canonical function calls are not shown. The hierarchy of calls from other top-level kernel functions is shown in Figure 2.

- **nml_interp_block_delete**
- **nml_interp_close**
  - **reset_interp_wm**
- **nml_interp_execute**
  - execute_block
  - read_line
  - read_text
  - write_g_codes
  - write_m_codes
- **nml_interp_exit**
  - **reset_interp_wm**
- **nml_interp_halt**
- **nml_interp_init**
  - ***CANON_VECTOR::CANON_VECTOR***
  - ***GET_B_ORIGIN***
  - ***GET_LENGTH_UNITS***
  - ***GET_ORIGIN***
  - ***GET_PLANE***
  - ***GET_PROGRAM_ORIGIN***
  - ***INIT_CANON***
  - *io_wm_flood*
  - *io_wm_mist*
  - *io_wm_speed*
  - *io_wm_spindle*
  - *io_wm_tool*
  - *io_wm_tool_table*
  - *NML_ROTPOSE::NML_ROTPOSE*
  - **reset_interp_wm**
  - *traj_wm_feed*
  - *traj_wm_pos*
  - *traj_wm_traverse*
  - write_g_codes
  - write_m_codes
- **nml_interp_open**
- **nml_interp_optional_stop**
- **nml_interp_pause**
- **nml_interp_read**
  - read_line
  - read_text
  - **reset_interp_wm**
- **nml_interp_resume**
- **nml_interp_run**
- **nml_interp_single_block**
- **nml_interp_step**
- **read_keyboard_line**
  - close_and_downcase

Figure 4. Interpreter Interface Function Call Hierarchy

This shows the hierarchy of function calls from interface functions in driver.cc, to:

1. kernel functions defined in rs274kt.cc (shown in ordinary typeface).
2. world modeling functions which are externally defined when the interpreter is integrated with the rest of EMC, but are defined in driver.cc in the stand-alone (shown in *italic*)
3. other interface functions defined in rs274kt.cc (shown in **boldface**)
4. special functions (not canonical functions) defined in canon.cc (or, for CANON_VECTOR::CANON_VECTOR, in canon.hh) (shown in ***bold italic***).

Eight interface functions defined in rs274kt.cc but not used in the stand-alone are not shown.

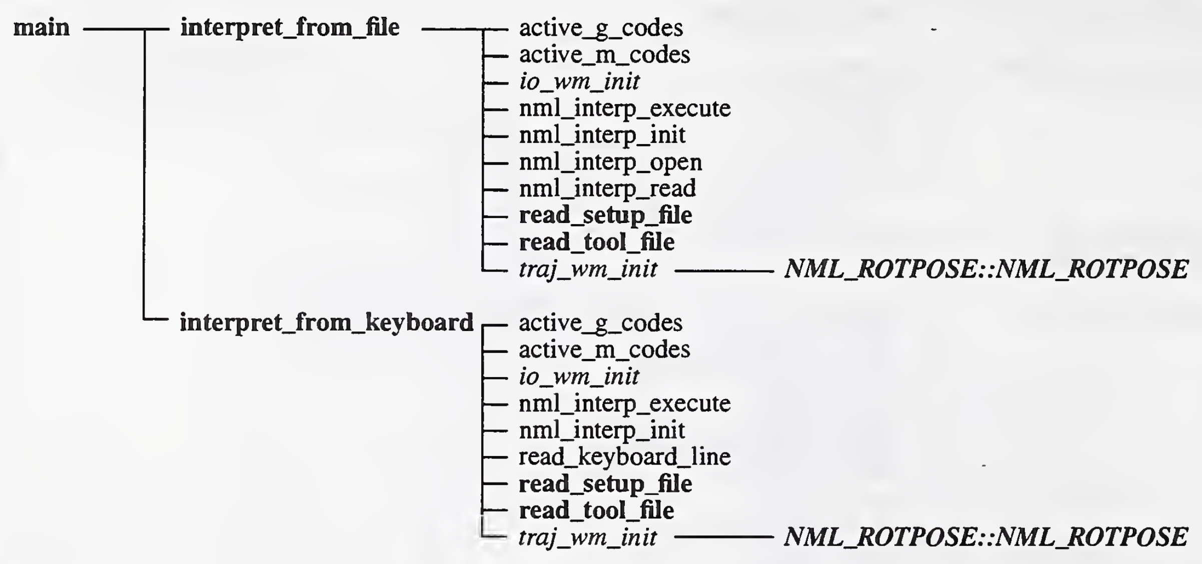

Figure 5. Interpreter Driver Function Call Hierarchy (for stand-alone interpreter)

This shows the hierarchy of function calls from main in the stand-alone driver file, driver.cc, to:

1. world modeling functions which are externally defined when the interpreter is integrated with the rest of EMC, but are defined in driver.cc in the stand-alone (shown in *italic*)

2. other functions defined in driver.cc (shown in **boldface**)

3. interface functions defined in rs274kt.cc (shown in ordinary typeface).

A.2 Source Code Documentation

The source code is heavily documented. In general, for each function, four fields are given:

1. Returned Value - a description of possible returned values and the circumstances in which particular values may be returned. In most kernel functions and many other functions, either OK or ERROR may be returned.

2. Side Effects - a description of the important side effects (things other than the returned value) of executing a function. Since the returned value of most functions is used to indicate error status, the side effects of most functions are important.

3. Called By - a list of functions which call the function being documented.

4. Argument Values - a one-line description of the meaning of each argument to a function, placed immediately after the declaration of the argument. This field is omitted if there are no arguments.

In addition to these four fields, most functions have a paragraph or two (up to several pages) of discussion. Where a function implements an algorithm for geometric or numerical calculation (as do many of the functions having to do with cutter radius compensation, for example), the algorithm is described. Many citations to specific pages of the K&T manual are included in these discussions.

Appendix B Functional Details

B.1 Error Handling and Exiting

The interpreter detects and flags most kinds of illegal input. Unreadable input, missing words, extra words, out-of-bounds numbers, and illegal combinations of words, for example, are all detected. The interpreter does not check for axis overtravel or excessively high feeds or speeds, however. The interpreter also does not detect situations where a legal command does something unfortunate, such as machining a fixture.

Error handling and exiting are handled together 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 error message is generated for each error. There are three sets of error messages:

1. input errors detected in the kernel or interface functions.
2. input errors detected in the stand-alone driver.
3. interpreter bugs detected in the kernel or interface functions (which should never occur).

The three sets of error messages are listed in the three subsections of Appendix D. Each message is intended to explain the error that triggered it clearly enough that a machine operator or NC programmer will be able to understand it and locate the error in the input. In the stand-alone interpreter, the line of NC code that caused the error will be printed, if the code is coming from a file. If the code is coming from the keyboard, it is already printed when the error is detected.

If an error occurs, or if it is time to exit, control is passed back up through the function call hierarchy to some driver function (for the stand-alone) or to an EMC function (for the integrated interpreter). Section 2.1.1 describes interpreter behavior in case of an error.

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.

Reporting errors detected in the kernel or interface functions (set 1 above) is handled entirely by a compiler macro called ERROR_MACRO. The version of this macro currently in use must be a macro, not a function, because it contains a "return" statement. All it does is find the error number of the error, call the nml_log_error function, and return ERROR. The macro has a "name" argument, which is not currently being used. That argument is for the name of the function returning the error and was used in earlier versions of the interpreter. Different versions of nml_log_error are used the stand-alone and integrated interpreters. In the stand-alone, nml_log_error prints the error to stderr.

Where one function calls a second function that may detect and report an error, the first function uses a compiler macro called ERROR_MACRO_PASS. All it does is return ERROR. It does not need to report anything since the second function has already done the reporting.

Reporting errors which are interpreter bugs (set 3 above) is handled by the compiler macro BUG_MACRO. This behaves the same as ERROR_MACRO. However, all the messages which can be sent via the BUG_MACRO include the name of the function in which the bug occurred.

Reporting input errors in the driver is handled by the macro DRIVER_ERROR, which takes both a message string and a value as arguments, so that it can identify the error more precisely. It also prints messages to stderr and returns ERROR.

In the interpreter source code, error messages from the first and third sets appear both in the text of functions and in an array of error messages. Error messages in the first set have indexes from 1 to 199, while those in the second set have indexes from 200 to 299. The array is used so that error messages can be exported by number (the index number of the message in the array) to other parts of the EMC system. The receiver must have a copy of the array, of course. The text of the messages could be deleted from the source code, and only the array index used, but that would delete important self-documentation from the source code. Keeping the array and the source code consistent requires attention by the system programmer(s) if changes are made but has been largely automated. The interpreter will report "unspecified error" if an error occurs for which the message in the function does not match any message in the array. If "unspecified error" is ever received, that exposes the internal inconsistency.

B.2 Cyclic Operation

In the integrated interpreter, input may be taken from a file or via manual data input (MDI) from the controller console. In the stand-alone, input may be taken from a file or from the keyboard of the computer running the interpreter. In all cases, two major phases of interpretation take place. The first phase consists of reading a line of code, checking it somewhat, and storing the information from the line in a structure called a "block". The second phase, which is always started by a call to nml_interp_execute, consists of examining the block, checking it further, and making calls to canonical functions. With MDI input the nml_interp_execute function triggers both phases. With file input, in both the stand-alone and the integrated version, the first phase is started by a call to nml_interp_read. In the stand-alone with keyboard input, the first phase is started by a call to read_keyboard_line.

B.2.1 Read, Store, and Check

The software reads lines of RS274KT code one at a time. First the line is read into a buffer. Next, 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.

Each line of code is stored until it has been executed in a reusable "block" structure, which has a slot for every potential piece of information on the line. Each time a useful piece of information has been extracted from the line, the information is put into the block.

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 (check_g_codes, check_m_codes, and check_other_codes) are run. These functions look for logical errors, such as all axis words missing with a G code for motion in effect.

B.2.2 Execute

Once a block is built and checked, the block is executed by the execute_block function, which calls one or more functions named convert_XXXX. In RS274KT, as in all dialects of RS274, a line 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 the manual.

In the interpreter, items are executed in the order shown in Table 9 if they occur in the same block. Many of the kernel functions rely implicitly on this order being used. The source code documentation does not generally call out the fact that some other operation must have occurred before the one being documented.

1. comment (includes message).
2. set feed mode (G93, G94 — inverse time or per minute).
3. set feed rate (F).
4. set spindle speed (S).
5. select tool (T).
6. change tool (M6).
7. spindle on or off (M3, M4, M5).
8. coolant on or off (M7, M8, M9).
9. B-axis clamping on or off (M26, M27).
10. enable or disable overrides (M48, M49).
11. set tool length offset (P).
12. dwell (G4).
13. set active plane (G17, G18, G19).
14. cutter radius compensation on or off (G40, G41, G42)
15. set length units (G70, G71)
16. set distance mode (G90, G91).
17. reset position of the origin (G98, G99).
18. perform motion (G0 to G3, G80 to G89).
19. stop (M0, M1, M2, M30, M60).

Table 9. Order of Execution

B.3 Tool Change

A CHANGE_TOOL canonical command call is made when an M6 code occurs on a line. Since this is a canonical command, it is not specialized to a particular machine. When the tool change is complete, the following conditions should prevail.

- The spindle should be stopped.
- The tool that was selected (by a T word on the same line or on any line after the previous tool change) should be in the spindle. The T number is an integer giving the id of the tool, not its changer slot.

- If the selected tool was not in the spindle before the tool change, the tool that was in the spindle (if there was one) should be in its changer slot.
- The coordinate axes should be stopped in the same absolute position they were in before the tool change (but the spindle may be re-oriented).
- No other changes should be made.

The tool change may include axis motion while it is in progress. For any machine, this motion must be predictable. It is up to the system programmers and operators to ensure that any such motion will not damage the machine or the workpiece.

If the tooling data is incorrect, so that the tool that was supposed to be selected is not actually in the expected slot, the tool change operation is not expected to be able to detect this, and no error will necessarily result. If the machine is able to detect this error, however, it should be detected and program execution should stop.

B.4 Tool Length Offsets

Tool length offsets are given as positive numbers in the tool table. Using a tool length offset is programmed using P with the desired table index. Using no tool length offset is programmed using P0 [KT, pages 4-18, 4-19]. The P number is checked for being a non-negative integer when it is read. The interpreter behaves as follows.

1. If a positive P number is programmed:

a. If the P number is the same as the slot number of the tool currently in the spindle, a USE_TOOL_LENGTH_OFFSET(length) function call is made (where length is the value of the tool length offset entry in the tool table whose index is the P number), tool_length_offset is reset in the machine settings model, and the value of current_z in the model is adjusted.

b. If the P number is not the same as the slot number of the tool currently in the spindle, an error message is printed, and ERROR is returned.

2. If P0 is programmed, USE_TOOL_LENGTH_OFFSET(0.0) is called, tool_length_offset is reset to 0.0 in the machine settings model, and the value of current_z in the model is adjusted.

B.5 Cutter Radius Compensation

The complete set of canonical functions includes functions which turn cutter radius compensation[1] on and off, so that cutter radius compensation can be performed in the controller executing the canonical functions. In the RS274KT interpreter, however, these commands are not used. Compensation is done by the interpreter and reflected in the output commands, which continue to direct the motion of the center of the cutter tip. This simplifies the job of the motion controller while making the job of the interpreter a little harder.

The algorithms for the first and last moves of cutter radius compensation used in the interpreter differ from those in the manual [K&T, pages 4-21 to 4-61]. The behavior for the intermediate moves is the same, except that some situations treated as errors in the interpreter are not treated as errors in the manual. In particular, the interpreter treats concave corners and concave arcs into which the tool will not fit as errors, while the manual does not. See Figure 6.

1. The term "cutter diameter compensation" is often used to mean the same thing.

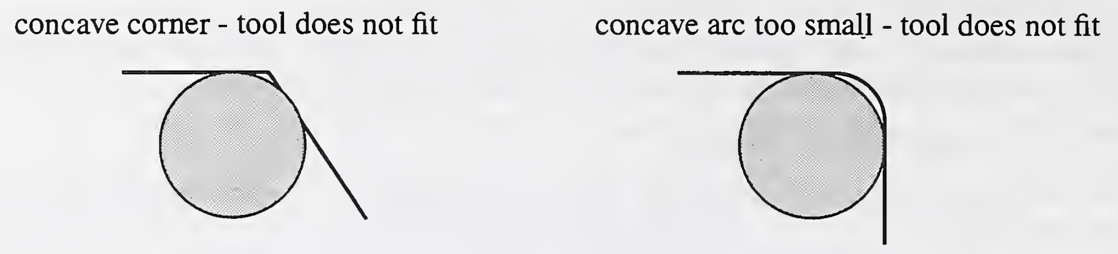
concave corner - tool does not fit
concave arc too small - tool does not fit

Figure 6. Two Cutter Radius Compensation Errors

In both examples, the line represents a tool path, and the circle represents the cross section of a tool following the path using cutter radius compensation (tangent to one side of the path.)

The interpreter machine model keeps three data items for cutter radius compensation: the setting itself (right, left, or off), program_x, and program_y. The last two represent the X and Y positions which are given in the NC code while compensation is on. When compensation is off, these both are set to a very small number (10^{-20}) whose symbolic value (in a #define) is "unknown". The interpreter machine model uses the data items current_x and current_y to represent the position of the center of the tool tip (in the currently active coordinate system) at all times.

The algorithm used for the first move when the first move is a straight line is to draw a straight line from the destination point which is tangent to a circle whose center is at the current point and whose radius is the radius of the tool. The destination point of the tool tip is then found as the center of a circle of the same radius tangent to the tangent line at the destination point. This is shown in Figure 7, which shows the construction in the XY plane. If the programmed point is inside the initial cross section of the tool (the circle on the left), an error is signalled.

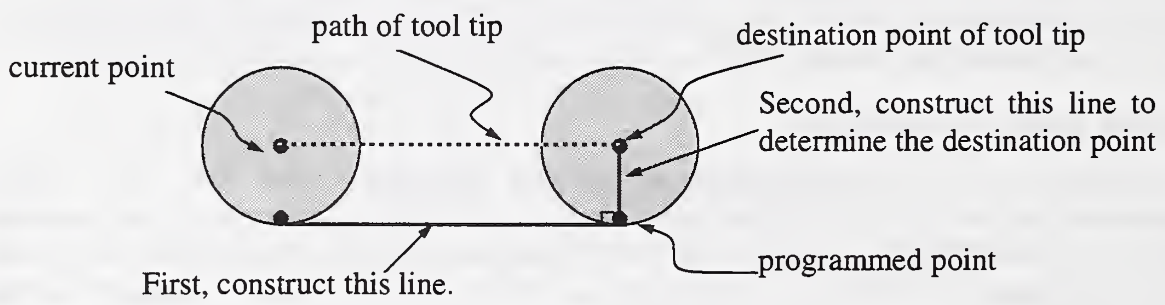
path of tool tip
destination point of tool tip
current point
Second, construct this line to determine the destination point
programmed point
First, construct this line.

Figure 7. First Cutter Radius Compensation Move - Straight

If the first move after cutter radius compensation has been turned on is an arc, the arc which is generated is derived from an auxiliary arc which has its center at the programmed center point, passes through the programmed end point, and is tangent to the cutter at its current location. If the auxiliary arc cannot be constructed, an error is signalled. The generated arc moves the tool so that it stays tangent to the auxiliary arc throughout the move. This is shown in Figure 8.

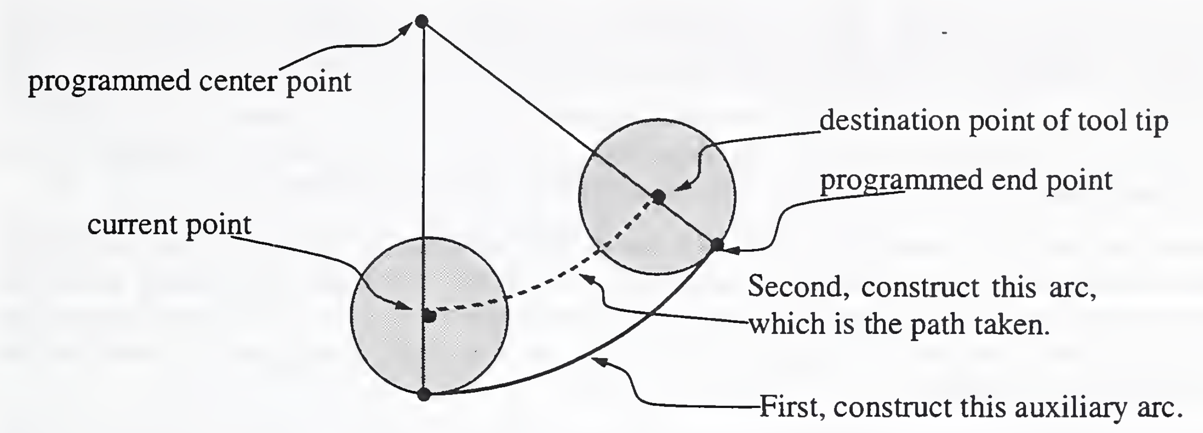
programmed center point
destination point of tool tip
programmed end point
current point
Second, construct this arc, which is the path taken.
First, construct this auxiliary arc.

Figure 8. First Cutter Radius Compensation Move - Arc

If the first move is either a straight line or an arc with the XY plane selected, the Z axis may also move at the same time. It will move linearly, as it does when cutter radius compensation is not being used.

After the first move of cutter radius compensation, the interpreter keeps the tool tangent to the programmed path on the appropriate side. If a convex corner is on the path, an arc is inserted to go around the corner. The radius of the arc is the tool radius.

When cutter radius compensation is turned off, no special exit move takes place. The next move is what it would have been if cutter radius compensation had never been turned on and the previous move had placed the tool at its current position.

If the interpreter has been compiled with the "DEBUG" option on, the interpreter signals when cutter radius compensation is turned on or off by calling the COMMENT canonical function with a message to that effect.

B.6 Inverse Time Feed Rate

In the RS274KT language, two feed modes are recognized: units per minute [K&T, pages 3.8 - 3.9] and inverse time [K&T, pages 3.9 - 3.18]. Programming G94 starts the units per minute mode. Programming G93 starts the inverse time mode.

In units per minute feed mode, an F word (no, not *that* F word; we mean *feedrate*) is interpreted to mean the tool tip should move at a certain number of inches per minute, millimeters per minute, or degrees per minute, depending upon what length units are being used and which axis or axes are moving.

In inverse time feed mode, an F word is interpreted to mean the move should be completed in [one divided by the F word] minutes. For example, if the F word is 2.0, the move should be completed in half a minute.

The interpreter handles the inverse time feed rate mode internally. The canonical functions have no command for putting a machine into this mode. If the interpreter is compiled with DEBUG #defined, when the interpreter converts a G93 or G94 command, it prints a comment saying it is going into inverse time feed mode (if G93) or into units per minute feed mode (if G94).

As specified in the manual, when the interpreter is in inverse time feed mode, an F word must appear on every line which has a motion. For each programmed motion (G1, G2, or G3), the interpreter calculates what the feed rate must be in units per minute to accomplish the move in the specified time. A SET_FEED_RATE canonical command (which always means units per minute) is output by the interpreter with this calculated rate. The calculation is: multiply the input F word by the path length of tool tip motion appropriate to the given kind of motion.

Canned cycles G81 through G89 cannot be run in inverse time feed mode. The manual [K&T, page 4.63] says the control should automatically be put into units per minute feed mode for canned cycles, but in the interpreter, a change to the units per minute feed mode must be done by an explicit G94 command. An error will occur in the interpreter if a canned cycle is called while in inverse time feed mode.

Cutter radius compensation cannot be used in inverse time feed mode. An error will occur in the interpreter if this is attempted in the input code.

Being in inverse time feed mode does not affect G0 (rapid traverse) motions. It is still necessary for F0 to be in effect for G0 to run without error.

Appendix C Transcript of a Session

This is a transcript of a session using the stand-alone interpreter with keyboard input. Characters entered by the user are shown in **boldface**. All user input is followed by a carriage return not shown here.

```
1} rs274kt
 1 N0 SET_FEED_REFERENCE(CANON_XYZ)
name of tool file => ../tool/gm_all
name of setup file => setup/test1
READ => g1 x3 y1 f20.0
EXEC <-;
 2 N ... SET_FEED_RATE(20.0000)
 3 N ... STRAIGHT_FEED(3.0000, 1.0000, 0.0000, 0, 0.0000)
READ => g2 x3 g91 i1.5 z0.5
EXEC <-;
 4 N ... COMMENT("interpreter: distance mode changed to incremental")
 5 N ... ARC_FEED(6.0000, 1.0000, 4.5000, 1.0000, -1, 0.5000, 0,0.0000)
READ => (that was a helical arc)
EXEC <-;
 6 N ... COMMENT("that was a helical arc")
READ => t2
EXEC <-;
interpreter error 91: Unknown tool_id used
READ => m6 t1
EXEC <-;
 7 N ... SELECT_TOOL(61)
 8 N ... CHANGE_TOOL(61)
READ => n15 g90 m3 p61 x7 y3 g81 r0.2 z-0.6676 s2000 m7
EXEC <-;
 9 N15 SET_SPINDLE_SPEED(2000.0000)
 10 N15 START_SPINDLE_CLOCKWISE()
 11 N15 MIST_ON()
 12 N15 USE_TOOL_LENGTH_OFFSET(0.0000)
 13 N15 COMMENT("interpreter: distance mode changed to absolute")
 14 N15 STRAIGHT_TRAVERSE(7.0000, 3.0000, 0.5000, 0, 0.0000)
 15 N15 STRAIGHT_TRAVERSE(7.0000, 3.0000, 0.2000, 0, 0.0000)
 16 N15 STRAIGHT_FEED(7.0000, 3.0000, -0.6676, 0, 0.0000)
 17 N15 STRAIGHT_TRAVERSE(7.0000, 3.0000, 0.2000, 0, 0.0000)
READ => m2
EXEC <-;
 18 N ... STOP_SPINDLE_TURNING()
 19 N ... MIST_OFF()
 20 N ... PROGRAM_END()
READ => quit
```

Appendix D Error Messages

The error messages used throughout the interpreter are intended to be self-explanatory. There are three categories of error message: interpreter kernel and interface input error messages, interpreter kernel internal error messages, and interpreter driver input error messages. These are described in Appendix D.1, Appendix D.2, and Appendix D.3, respectively. Each list is arranged alphabetically (except the "unspecified error" message is first in Appendix D.1). Messages are in **boldface** type. Following each message in Appendix D.1 and Appendix D.3 is the name of the function or functions in which it is found, printed in *italics*.

D.1 Interpreter Kernel and Interface Input Error Messages

This is a list of all input error messages in the interpreter kernel and interface. Each message describes some kind of error in the input to the interpreter. Each message has a number, which is the index number of the message in the array of error messages in the source code; this number is printed at the beginning of each line.

The "Unspecified error" error message should never appear. If it does, this means there is an error message in the source code which has been left out of the error message array. If this happens, please contact kramer@cme.nist.gov by email; include the message and describe the circumstances in which it appeared.

0. Unspecified error	see note above
1. B-axis value greater than 359.999	*read_b*
2. Bad character used	*read_one_item*
3. Bad format real number	*read_real*
4. Bad format unsigned integer	*read_integer_unsigned*
5. Bad tool radius value with cutter radius comp	*convert_arc_comp1, convert_arc_comp2, convert_straight_comp1, convert_straight_comp2*
6. Bad value for pitch	*arc_data_pitch*
7. Cannot change origin with canned cycle	*convert_axis_offsets*
8. Cannot change origin with cutter radius comp	*convert_axis_offsets*
9. Cannot change units with cutter radius comp	*convert_length_units*
10. Cannot do G1 with zero feed rate	*convert_straight*
11. Cannot do canned cycle in incremental distance mode	*convert_cycle*
12. Cannot do canned cycle in inverse time feed mode	*convert_cycle*
13. Cannot do canned cycle with zero feed rate	*convert_cycle*
14. Cannot have concave corner with cutter radius comp	*convert_straight_comp2*
15. Cannot make arc with zero feed rate	*convert_arc*
16. Cannot perform G84 unless spindle is turning	*convert_cycle_g84*
17. Cannot put a B-axis command in a canned cycle	*check_other_codes*
18. Cannot turn cutter radius comp on out of XY-plane	*convert_cutter_compensation_on*
19. Cannot turn cutter radius comp on when already on	*convert_cutter_compensation_on*
20. Cannot use G0 with cutter radius comp	*convert_straight*
21. Cannot use M25 in incremental distance mode	*check_m_codes*
22. Cannot use XZ plane with cutter radius comp	*convert_set_plane*
23. Cannot use YZ plane with cutter radius comp	*convert_set_plane*
24. Cannot use a G code for motion with G4	*check_g_codes*

25. Cannot use a G code for motion with G98	*check_g_codes*
26. Cannot use axis commands with G4	*check_g_codes*
27. Cannot use inverse time feed with cutter radius comp	*convert_arc, convert_straight*
28. Command too long	*close_and_downcase, nml_interp_execute*
29. Concave corner with cutter radius comp	*convert_arc_comp2*
30. Coordinate setting given with G80	*convert_motion*
31. Cutter gouging with cutter radius comp.	*convert_straight_comp1*
32. D word missing with cutter radius comp on.	*convert_cutter_compensation_on*
33. D word on line with no cutter comp on (G41 or G42) command	*check_other_codes*
34. Dwell time missing with G4	*check_g_codes*
35. E word missing in G83 or G87 canned cycle	*convert_cycle*
36. E word on line with no X, Y, or Z	*check_other_codes*
37. E word used without G83 or G87	*check_other_codes*
38. F word missing with inverse time G1 move	*convert_straight*
39. F word missing with inverse time arc move	*convert_motion*
40. Feed rate must be zero to use G0.	*convert_straight*
41. File ended with no stopping command given	*read_text*
42. G code out of range.	*read_g*
43. I and J words missing for arc in XY-plane.	*convert_arc*
44. I and K words missing for arc in XZ-plane	*convert_arc*
45. I word on line with no X, Y, or Z.	*check_other_codes*
46. J and K words missing for arc in YZ-plane	*convert_arc*
47. J word on line with no X, Y, or Z.	*check_other_codes*
48. K word on line with no X, Y, or Z	*check_other_codes*
49. Line number greater than 99999.	*read_line_number*
50. M code greater than 99.	*read_m*
51. Multiple B words on one line	*read_b*
52. Multiple D words on one line.	*read_d*
53. Multiple E word settings on one line.	*read_delta*
54. Multiple F words on one line	*read_f*
55. Multiple I words on one line	*read_i*
56. Multiple J words on one line	*read_j*
57. Multiple K words on one line.	*read_k*
58. Multiple P word tool length offsets on one line	*read_tool_length_offset*
59. Multiple R words on one line.	*read_r*
60. Multiple S word spindle speed settings on one line	*read_spindle_speed*
61. Multiple T words on one line	*read_tool_id*
62. Multiple X words on one line.	*read_x*
63. Multiple Y words on one line.	*read_y*
64. Multiple Z words on one line	*read_z*
65. Must use G0 with M25	*check_m_codes*
66. Negative F word found	*read_f*
67. Negative or zero I word for arc in YZ-plane	*convert_arc*
68. Negative or zero J word for arc in XZ-plane	*convert_arc*
69. Negative or zero K word for arc in XY-plane.	*convert_arc*
70. Negative spindle speed found.	*read_spindle_speed*

71. Nested comment found	*close_and_downcase*
72. No digits found while reading number	*read_real*
73. Null missing after newline	*close_and_downcase*
74. Percent sign not first item on first line	*nml_interp_open*
75. R clearance plane unspecified in canned cycle	*convert_cycle*
76. R value less than Z value in canned cycle	*convert_cycle*
77. R word on line with no X, Y, or Z	*check_other_codes*
78. R word used outside a canned cycle	*check_other_codes*
79. Radius to end of arc differs from radius to start of arc	*arc_data, arc_data_comp*
80. Too many G codes on line	*check_g_codes*
81. Too many M codes on line	*check_m_codes*
82. Tool index out of bounds	*convert_tool_length_offset*
83. Tool offset index differs from current slot	*convert_tool_length_offset*
84. Tool radius not less than arc radius with cutter radius comp	*convert_arc_comp2*
85. Two G codes used from same modal group	*read_g*
86. Two M codes used from same modal group	*read_m*
87. Unable to open file	*nml_interp_open*
88. Unclosed comment found	*close_and_downcase*
89. Unknown G code used	*read_g*
90. Unknown M code used	*read_m*
91. Unknown tool_id used	*convert_tool_select*
92. X and Y words missing for arc in XY-plane	*convert_arc*
93. X and Z words missing for arc in XZ-plane	*convert_arc*
94. X missing for arc with I word in YZ-plane	*convert_arc*
95. X, Y, Z, and B words all missing with G0 or G1	*convert_straight*
96. Y and Z words missing for arc in YZ-plane	*convert_arc*
97. Y missing for arc with J word in XZ-plane	*convert_arc*
98. Z missing for arc with K word in XY-plane	*convert_arc*
99. Z value unspecified in canned cycle	*convert_cycle*
100. Zero or negative E word found	*read_delta*

D.2 Interpreter Kernel Internal Error Messages

The following error messages should never appear but are provided as a check on the internal workings of the interpreter kernel. Each message has a number, which is the index number of the message in the array of error messages in the source code; this number is printed here at the beginning of each line. The name of the function in which the error has occurred is part of each message. The appearance of one of these means there is a bug in the source code for the interpreter. If one of these error messages ever appears, please contact kramer@cme.nist.gov by email; include the message and describe the circumstances in which it appeared.

200. B_turn setting not -1, 0, or 1 in utility_enhance_block
201. Code is not G0 or G1 in convert_straight
202. Code is not G0 to G3 or G80 to G89 in convert_motion
203. Code is not G17, G18, or G19 in convert_set_plane
204. Code is not G2 or G3 in arc_data
205. Code is not G2 or G3 in arc_data_comp
206. Code is not G40, G41, or G42 in convert_cutter_compensation
207. Code is not G70 or G71 in convert_length_units
208. Code is not G90 or G91 in convert_distance_mode
209. Code is not G93 or G94 in convert_feed_mode
210. Code is not G98 or G99 in convert_axis_offset
211. Code is not M0, M1, M2, M30 or M60 in convert_stop
212. Cycle code not G81 to G89 in convert_cycle
213. Plane is not XY, YZ, or XZ in convert_arc
214. Read_b should not have been called
215. Read_comment should not have been called
216. Read_d should not have been called
217. Read_delta should not have been called
218. Read_f should not have been called
219. Read_g should not have been called
220. Read_i should not have been called
221. Read_j should not have been called
222. Read_k should not have been called
223. Read_line_number should not have been called
224. Read_m should not have been called
225. Read_r should not have been called
226. Read_spindle_speed should not have been called
227. Read_tool_id should not have been called
228. Read_tool_length_offset should not have been called
229. Read_x should not have been called
230. Read_y should not have been called
231. Read_z should not have been called
232. Side fails to be right or left in convert_straight_comp1
233. Side fails to be right or left in convert_straight_comp2
234. Sscanf failure in read_integer_unsigned
235. Sscanf failure in read_real

D.3 Interpreter Driver Input Error Messages

The following error messages may be printed by the stand-alone interpreter but not the integrated interpreter. Each message describes an error in the input to the stand-alone interpreter that is not an interpreter kernel error. The messages originate in the source code for the stand-alone interpreter driver. A word in ***bold italics*** stands for a number or word that will differ according the exact nature of the error. The error messages are not kept in an array.

1. **Bad input line "*text_line*" in setup file** . . . . *read_setup_file*
2. **Bad input line "*text_line*" in tool file**. . . . . *read_tool_file*
3. **Bad setup file format**. . . . . *read_setup_file*
4. **Bad tool file format** . . . . *read_tool_file*
5. **Bad value *given_value* for block_delete in setup file** . . . . *read_setup_file*
6. **Bad value *given_value* for cutter_radius_comp in setup file**. . . . . *read_setup_file*
7. **Bad value *given_value* for distance_mode in setup file** . . . . *read_setup_file*
8. **Bad value *given_value* for feed_mode in setup file**. . . . . *read_setup_file*
9. **Bad value *given_value* for flood in setup file**. . . . . *read_setup_file*
10. **Bad value *given_value* for length_units in setup file** . . . . *read_setup_file*
11. **Bad value *given_value* for mist in setup file** . . . . *read_setup_file*
12. **Bad value *given_value* for plane in setup file** . . . . *read_setup_file*
13. **Bad value *given_value* for speed_feed_mode in setup file**. . . . . *read_setup_file*
14. **Bad value *given_value* for spindle_turning in setup file** . . . . *read_setup_file*
15. **Cannot open *file_name*** . . . . *read_setup_file, read_tool_file*
16. **Unknown attribute *attribute_name* in setup file**. . . . . *read_setup_file*
17. **Usage "rs274kt" or "rs274kt filename" or "rs274kt filename continue"** . . . . *main*

Appendix E Production Rules for Line Grammar and Syntax

The following is a production rule definition of what the RS274KT interpreter recognizes as valid combinations of symbols which form a readable line (the top of this production hierarchy). The productions are arranged alphabetically to make connections easy to trace.

The productions are intended to be unambiguous. That is, no permissible line of code can be interpreted more than one way. To make the productions below entirely unambiguous, it is implicit that if a string of characters that can meet the requirements of an ordinary comment also can be interpreted as a message, that string is a message and not an ordinary comment.

The term comment_character is used in the productions but not defined there. A comment character is any printable character plus space and tab, except for a left parenthesis or right parenthesis. This implies comments cannot be nested.

It is implicit in the production rules that, except inside parentheses, space and tab characters may be ignored.

These production rules do not include constraints implied by the semantics of the interpreter. Most of the constraints are in terms of combinations of words (as defined below). Many lines of code that are readable under these production rules will not be executable because they violate constraints. For example G200 would be a valid G word under the production rules, but it does not satisfy the constraint that the unsigned integer after the G may not be 200. Any constraint violation will be detected by the interpreter and will result in an error message. The error messages are intended to describe the constraint which has been violated and are included in Appendix D

In Section 5.2 the term "number" is defined. That term is defined in the productions as either "unsigned_integer" or "real_number", but "number" is not used by any other rule. The RS274KT language does not use signed integers.

E.1 Production Language

The symbols in the productions are mostly standard syntax notation. Meanings of the symbols follow.

=	The symbol on the left of the equal sign is equivalent to the expression on the right
+	followed by
I	or
.	end of production (a production may have several lines)
[ ]	zero or one of the expression inside square brackets may occur
{ }	zero to many of the expression inside curly braces may occur
( )	exactly one of the expression inside parentheses must occur

E.2 Productions

Any term in this subsection that is used on the right of an equal sign but is not defined in this subsection (i.e., does not appear on the left in any definition) is defined in the next subsection in terms of characters.

argument_word = (letter_b | letter_e | letter_i | letter_j | letter_k | letter_r | letter_x |
letter_y | letter_z) + real_number .
comment = message | ordinary_comment .
comment_character = *see explanation above* .
digit = zero | one | two | three | four | five | six | seven | eight | nine .
g_word = letter_g + unsigned_integer .
letter_b = big_b | little_b .
letter_d = big_d | little_d .
letter_e = big_e | little_e .
letter_f = big_f | little_f .
letter_g = big_g | little_g .
letter_i = big_i | little_i .
letter_j = big_j | little_j .
letter_k = big_k | little_k .
letter_m = big_m | little_m .
letter_n = big_n | little_n .
letter_p = big_p | little_p .
letter_r = big_r | little_r .
letter_s = big_s | little_s .
letter_t = big_t | little_t .
letter_x = big_x | little_x .
letter_y = big_y | little_y .
letter_z = big_z | little_z .
line = [block_delete] + [line_number] + {segment} + end_of_line .
line_number = letter_n + digit + [digit] + [digit] + [digit] + [digit] .
message = left_parenthesis + {white_space} + letter_m + {white_space} + letter_s +
{white_space} + letter_g + {white_space} + comma + {comment_character} +
right_parenthesis .
m_word = letter_m + unsigned_integer .
number = real_number | unsigned_integer .
ordinary_comment = left_parenthesis + {comment_character} + right_parenthesis .
rate_word = (letter_f | letter_s) + real_number .
real_number = [plus | minus] +
((digit + { digit } + [decimal_point] + {digit}) | (decimal_point + digit + {digit})) .
segment = word | comment .
tool_word = (letter_d | letter_p | letter_t) + unsigned_integer
unsigned_integer = digit + { digit } .
white_space = { tab | space }
word = g_word | m_word | tool_word | rate_word | argument_word .

E.3 Production Tokens in Terms of Characters

We have omitted the letters and digits in the list below, since they are all the obvious single characters. For example, one is ‘1’, big_b is ‘B’, and little_b is ‘b’. The list should be used as if these obvious items were included. Note that not every letter of the alphabet is included (A, C, H, L, O, Q, U, V, and W are omitted).

block_delete = ‘/’
comma = ‘,’
decimal_point = ‘.’
end_of_line = ‘ ’ (non-printable newline character)
left_parenthesis = ‘(’
minus = ‘-’
plus = ‘+’
right_parenthesis = ‘)’
space = ‘ ’ (non-printable space character)
tab = ‘ ’ (non-printable tab character)

Appendix F Setup File Format

The format of a setup file is shown in Table 10. A setup file is used by giving its name when prompted to do so when the interpreter starts up, as described in Section 4.1.

The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The interpreter just skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in Table 10 describes the data columns, so it is suggested (but not required) that that line always be included in the header.

The interpreter reads only the first two columns of the table. The third column, “Other Possible Values,” is included here for information.

Each line of the file contains the name of an attribute in the first column and the value to which that attribute should be set in the second column. Attribute names must be spelled exactly as shown in Table 10 in lower case letters. Where the value is shown in upper case letters in the table, upper case letters must be used, and the alternative values are also required to be in upper case letters. The same attribute name should not be used twice, but the interpreter does not check that. If any attribute name not given in the table is used, an error will result.

The lines do not have to be in any particular order. Switching the order of lines has no effect on the interpreter or on how any NC program will be executed (unless the same attribute is used on two or more lines, which should not normally be done, in which case the data for only the last such line will persist).

Attribute	Value	Other Possible Values
axis_offset_b	0.0	any real number
axis_offset_x	0.0	any real number
axis_offset_y	0.0	any real number
axis_offset_z	0.0	any real number
block_delete	ON	OFF
current_b	0.0	any real number
current_x	0.0	any real number
current_y	0.0	any real number
current_z	0.0	any real number
cutter_radius_comp	OFF	LEFT, RIGHT
cycle_e	0.1	any positive real number
cycle_r	0.0	any real number
cycle_z	0.0	any real number not less than cycle_r
distance_mode	ABSOLUTE	INCREMENTAL
feed_mode	PER_MINUTE	INVERSE_TIME
feed_rate	5.0	any positive real number
flood	OFF	ON
length_units	MILLIMETERS	INCHES
mist	OFF	ON
motion_mode	80	0,1,2,3,81,82,83,84,85,86,97,88,89
plane	XY	YZ, ZX
slot_for_length_offset	1	any unsigned integer less than 69
slot_for_radius_comp	1	any unsigned integer less than 69
slot_in_use	1	any unsigned integer less than 69
slot_selected	1	any unsigned integer less than 69
speed_feed_mode	INDEPENDENT	SYNCHED
spindle_speed	1000.0	any non-negative real number
spindle_turning	STOPPED	CLOCKWISE, COUNTERCLOCKWISE
tool_length_offset	0.0	any non-negative real number
traverse_rate	199.0	any positive real number

Table 10. Sample Setup File

Appendix G Tool File Format

The format of a tool file is shown in Table 11. A tool file is used by giving its name when prompted to do so when the interpreter starts up, as described in Section 4.1.

The file consists of any number of header lines, followed by one blank line, followed by any number of lines of data. The interpreter just skips over the header lines. It is important that there be exactly one blank line (with no spaces or tabs, even) before the data. The header line shown in Table 11 describes the data columns, so it is suggested (but not required) that that line always be included in the header.

Each data line of the file contains the data for one tool. Each line has six entries, the first four of which are required, and the last two of which are optional. It makes reading easier if the entries are arranged in columns, as shown in the table, but the only format requirement is that there be at least one space or tab after each of the first four entries on a line. The meanings of the columns and the type of data to be put in each are as follows.

The "POCKET" column contains an unsigned integer which represents the pocket number (slot number) of the tool changer pocket in which the tool is placed. The entries in this column must all be different.

The "FMS" column contains an unsigned integer which represents a code number for the tool. The user may use any code for any tool, as long as the codes are unsigned integers. It is legal, but usually not a good idea, to have two tools with the same code number. Tools are selected by using this number.

The "TLO" column contains a real number which represents the tool length offset. This number will be used if tool length offsets are being used and this pocket is selected. This is normally a positive real number, but it may be zero or any other number if it is never to be used.

The "DIAM" column contains a real number which represents the diameter of the tool. This number is used only if tool diameter compensation is turned on using this pocket. This is normally a positive real number, but it may be zero or any other number if it is never to be used.

The "HOLDER" column may optionally be used to describe the tool holder. Any type of description is OK. This column is for the benefit of human readers only.

The "TOOL DESCRIPTION" column may optionally be used to describe the tool. Any type of description is OK. This column is for the benefit of human readers only.

The interpreter only reads data from the first four columns of each line. The rest of the line is read but ignored.

The units used for the length and diameter of the tool may be in either millimeters or inches, but if the data is used by an NC program, the program must call out the correct G code (G70 or G71) for those units before the data is used. The table shows a mixture of types of units.

The lines do not have to be in any particular order. Switching the order of lines has no effect on the interpreter or on how any NC program will be executed (unless the same slot number is used on two or more lines, which should not normally be done, in which case the data for only the last such line will persist).

POCKET	FMS	TLO	DIAM	HOLDER	TOOL DESCRIPTION
1	1	1.0	0.25	14141	0.25" end mill
20	1419	4.299	1.0	0	1" carbide end mill
21	1025	8.34	0.5	drill chuck	1/2" spot drill short
32	1764	296.515	8.5	0	8.5mm drill
41	1237	228.360	10.0	0	10mm x 1.25 tap
60	71117	0	0	0	large chuck

Table 11. Sample Tool File