[redacted]

SOFTWARE BY WILSERV

KMMM PASCAL

LEVEL IV

COMPILER/TRANSLATOR

REFERENCE MANUAL

WILSERV INDUSTRIES
P. O. BOX 456
BELLMAWR, NEW JERSEY 08031
(609) 227-8696

TABLE OF CONTENTS

PLEASE READ THIS FIRST.....................................1

REGISTRATION FORM..2

GETTING STARTED..4

SECURITY KEY...5

SYSTEM OVERVIEW..6

RESERVED WORDS...8

PREDECLARED IDENTIFIERS...................................12

SUPPLIED FUNCTIONS..14

SUPPLIED PROCEDURES.......................................17

LIMITATIONS...19

INPUT/OUTPUT PROCEDURES...................................20

EXTERNAL FILE ACCESS......................................23

OUTPUT FIELD FORMATTING...................................25

RUN TIME CONSIDERATIONS...................................26

OPERATING INSTRUCTIONS....................................27

ERROR MESSAGES..32

DISKETTE COPY...34

FOREIGN DEVICES...35

REPORTING PROBLEMS..36

USER SUPPORT..37

LOADING PROGRAMS.................................APPENDIX A

FILE NAME SYNTAX.................................APPENDIX B

RESERVED WORDS...................................APPENDIX C

PRE-DEFINED IDENTIFIERS..........................APPENDIX D

BIBLIOGRAPHY

PLEASE READ THIS FIRST

First, please make sure you have all the parts of this package. There should be a diskette marked 'Software by Wilserv', a security key, this Compiler/Translator reference manual (37 pages, 4 appendices and a bibliography) and an Editor reference manual. A registration form is bound into the Compiler/Translator reference manual.

If anything is missing, you must obtain the missing item from the company from which you purchased the KMMM Pascal package. Contact us ONLY if you cannot obtain satisfaction from the dealer.

Next, please fill out and return the registration form. Instructions for filling out this form are in the next section of this manual.

Finally, use the Editor to examine the files named 'ADDENDA' and 'ERRATA'. If information contained in the 'ADDENDA' file superceeds any information contained in this manual, you should use your favorite color felt tip pen to correct your copy of this manual.

The 'ERRATA' file contains a list of the outstanding problems known to exist in KMMM Pascal. It also contains a history of solved problems and the names of the individuals who first reported these problems.

REGISTRATION FORM

The following page is a registration form that must be returned, in its entirety, to Wilserv Industries, if you wish to take advantage of the support services offered by Wilserv.

We would like to point out that it is not necessary to send money along with the registration form. The offer of a user library is just an incentive for you to register. Users who simply send the registration form in, without ordering the user library, will receive the same level of support as those users who do order. However, those who do order, will automatically receive the newest version of KMMM Pascal, if the version designation on the registration form indicates that you do not have the latest version.

To protect your privacy, we do not provide the names and addresses of our registered users to any other organization, public or private.

This registration form is very valuable. Registered users can obtain a new version of KMMM Pascal for as little as $1.50. Since the current retail price of KMMM Pascal is $99, that means the registration form is worth $97.50.

We accept the registration form as proof of purchase. To deter possible fraud, the registration form is individually printed and color coded. Therefore, a xerox copy will be returned and your request for registration denied.

When you call our service desk for assistance, you will be asked for the hand written serial number on your security key. This is a seven digit number that starts with the digit eight. We will then check our files to see if we have received your registration form. If not, you will be provided a minimum level of assistance and urged to send the form in as soon as possible. Subsequent requests for assistance will be declined until we receive the form.

Please print when filling in the blanks at the bottom of the form. Enter the name of your company only if it is part of the mailing address. Use the standard two character post office state abbreviation, if you know it. If you have the version for the C64, do NOT check either of the other possible responses on the 'Computer Model' line. You must fill in the serial number of your security key. If you are ordering the user library and enclosing a check, please do NOT staple it to the form. Finally, send in the entire form, NOT just the piece at the bottom.

GETTING STARTED

Before going any further, you should make a working copy of the enclosed distribution diskette. If you do not know how to copy a diskette, please follow the instructions in the section titled 'DISKETTE COPY'. After making this working copy, please store the distribution diskette in a safe, dry, cool, non-magnetic place so that you can make another working copy if you somehow destroy the first working copy.

To become familiar with the KMMM Pascal package, we suggest that you compile and translate some or all of the sample source files. This will also serve to verify that the working copy you've made has no extraneous bits in any of its files.

First, read the section titled 'SECURITY KEY' to insure that you attach the security key to your computer correctly. Next, read the first page of the section titled 'OPERATING INSTRUCTIONS'. Then, study the material presented in Appendix A. This Appendix briefly describes a disk support program that allows you to use a single character to load and run a program.

To begin, run the Compiler. Enter 'PSIEVE' as the file name. Respond to the next two prompts by simply pressing the return key. The Compiler will then display the source file on the screen as it generates a sort of shorthand, called P-code, in memory. After the display of the source file has ended, respond to the next three prompts by simply pressing the return key. The last depression of the return key will cause the Translator to automatically load into memory and execute. The Translator will then convert the P-code generated by the Compiler into 6502 machine language. When the Translator ends, the generated machine language program will be in memory, ready to be 'RUN' immediately or 'SAVEd' to cassette or diskette for future use.

SECURITY KEY

The little black box, supplied as a part of this package, will be referred to, in this manual, as a security key. A similar device, supplied with the Flex File package, is called a 'dongle', but we don't like that name.

This security key must be attached to your computer before attempting to run the Compiler or using the 'E' command available with the Editor/Compiler. If you neglect to attach the security key, there will be NO warning message displayed.

The security key attaches onto the cassette port of your computer, with the black box shape on the key pointing up. Do not attempt to attach the security key unless the power is off. If a cassette drive is currently attached, unplug the cassette drive plug, attach the security key and plug the cassette drive plug onto the security key. !

If you own a copy of Flex File and the Flex File security key is attached to the cassette port, then simply plug the enclosed security key onto the back of the Flex File security key. Although the two security keys are very similar in appearance, the Flex File security key can not be used as a substitute for the KMMM Pascal security key.

Any program generated by KMMM Pascal does NOT require that a security key be in place. However, the run time package portion of the program generated from your source file is covered by copyright. We grant permission for unlimited copying of the generated program provided you do NOT remove the copyright display and provided you DO credit KMMM Pascal as having been used to generate that program in any documentation (printed or machine-readable) that accompanies the program.

SYSTEM OVERVIEW

KMMM Pascal is a true compiler, that is, it generates machine language from a Pascal source file. At the time this manual is being prepared, it, and ZOOM Pascal, are the only versions of Pascal available for the Commodore C64.

KMMM Pascal is a subset of the Pascal described by Jensen and Wirth in their book, which is the 'bible' for Pascal. It is also a subset of UCSD Pascal, and supplies all of the string functions supplied by UCSD Pascal. See the following sections of this manual for a full discussion of the differences between KMMM Pascal and standard Pascal.

! ZOOM Pascal is a subset of KMMM Pascal. It came into being when Abacus Software agreed to take level III of KMMM Pascal, rewrite the instruction manual, leave out the Editor/Compiler and distribute the resulting package as ZOOM Pascal.

This manual is only a guide to use of KMMM Pascal and is not intended to teach the user how to write programs in Pascal.

KMMM Pascal requires at least 24K of RAM to run successfully. It also requires some form of external storage device, for source files and generated programs. This external device may be either a cassette or a diskette drive. Although KMMM Pascal will work with a cassette drive, use of a cassette drive is suitable only for small source or data files. A cassette should NOT be used for loading the programs that make up KMMM Pascal.

The programs which make up KMMM Pascal are totally compatible with Commodore BASIC. They are loaded into memory and executed as if they were BASIC programs, even though they are written in machine language. This is also true for the program generated from your source file.

There is a text editor, which is used to prepare or modify Pascal source files to be used as input to the Compiler. The Editor is described in a separate manual following this Compiler/Translator manual.

Actually, there are two versions of the Editor. The bigger of the two is the Editor/Compiler, which is the editor to be used in most cases. The Editor is supplied for use in editing source files which are too large for the Editor/Compiler.

The Editor/Compiler is so named because it is a combination of the syntax checking portion of the Compiler with the smaller Editor. It allows a source file being created or modified to be checked for proper syntax before being stored on cassette or diskette.

The Compiler reads a source file stored on cassette or diskette and generates an intermediate P-code file in memory. This P-code file is used as input to the Translator, which is loaded into memory by the Compiler. The Translator creates true machine code which may be executed immediately or stored on cassette or disk for later execution. The source for the Translator is actually coded in KMMM Pascal, and the Translator is output of this system.

RESERVED WORDS

This section lists all the reserved words defined by 'standard Pascal'. If KMMM Pascal does not deviate from 'standard Pascal', then the reserved word is merely noted as being standard. If differences exist between 'standard Pascal' and KMMM Pascal, then the reserved word is followed by a description of the differences.

The following reserved word must be the first non-comment element of a KMMM Pascal source file:

PROGRAM: KMMM Pascal requires that a space and an identifier follow the reserved word 'PROGRAM', as does 'standard Pascal'. KMMM Pascal will bypass any words or special characters that follow the program identifier, up to and including a semi-colon.

```
PROGRAM SAMP1;
PROGRAM SAMP2(INPUT,OUTPUT);
PROGRAM SAMP3('description',date);
```

The following six reserved words are discussed out of alphabetical order, because they serve as the starting point of the major sections of a Pascal block:

LABEL: Standard. However, KMMM Pascal will flag the reserved word 'GOTO' as an unimplemented statement.

CONST: Standard. In addition, KMMM Pascal will allow you to define hexadecimal constants. A dollar sign, followed by two hexadecimal digits, serves to declare a constant of type 'CHAR'. A dollar sign, followed by four hexadecimal digits, serves to declare a constant of type 'INTEGER'.

```
CLEARSCREEN=$93;  (* type is 'CHAR' *)
PETSCREEN=$8000;  (* type is 'INTEGER' *)
C64SCREEN=$0400;  (* type is 'INTEGER' *)
```

TYPE: Standard, except that the reserved word 'FILE' may not appear in a type declaration. There are also implementation dependent restrictions.

VAR: Standard, except that the component type for a 'FILE' declaration may only be of type 'CHAR' or 'RECORD'. There are also certain implementation dependent restrictions.

FUNCTION: Standard, except that only value and variable parameters may be passed as arguments to a function. In addition, KMMM Pascal allows the type of the value returned by the function to be of type 'STRING'.

PROCEDURE: Standard, except that only value and variable parameters may be passed as arguments to a procedure.

The following remaining reserved words are listed in alphabetical order:

AND: Standard.

ARRAY: Number of dimensions limited to two. Index type may only be an integer sub-range.

BEGIN: Standard.

CASE: Standard, except that this reserved word may not appear in a record declaration. In additional, KMMM Pascal allows the selector expression in a 'CASE' statement to be of type 'STRING'.

DIV: Standard.

DO: Standard.

DOWNTO: Standard.

ELSE: Standard. In addition, KMMM Pascal allows 'ELSE' to be used as a synonym for 'OTHERWISE' in a 'CASE' statement.

END: Standard.

FILE: See the sections of this manual that discuss input/output.

FOR: Standard.

GOTO: Will be flagged as an unimplemented statement.

IF: Standard.

IN: The operand following the word 'IN' may only be a literal.

This literal may only be of type 'CHAR' or of an enumerated type.

MOD: Standard. However, the standard is not defined when the value of the second operand is negative, so we followed the standard set by the Pascal for the VAX machines.

NIL: Standard.

NOT: Standard.

OF: Standard.

OR: Standard.

OTHERWISE: Standard (ISO). May only appear in a 'CASE' statement.

PACKED: Treated as a comment.

RECORD: Standard, except that the reserved word 'CASE' may not appear in a record declaration. Also, there are certain implementation dependent restrictions.

REPEAT: Standard.

SET: Will be flagged as an unimplemented data type.

THEN: Standard.

TO: Standard.

UNTIL: Standard.

WHILE: Standard.

WITH: Will be flagged as an unimplemented statement.

In addition to the above reserved words, KMMM Pascal still retains four reserved from its early days in the marketplace. These are:

CALL: This reserved word functions exactly like the 'SYS' verb in BASIC. The reserved word 'CALL' is followed by an integer expression enclosed in parentheses. The expression is the address of the machine language routine to be executed. The routine must terminate in an 'RTS' instruction.

MEM: This reserved word is a predeclared variable identifier that allows you to address the entire address space of your computer. The internal declaration is as follows:

```
MEM: ARRAY[0..65565] OF CHAR;
```

Therefore, the reserved word 'MEM' must always be followed by an integer expression enclosed in brackets. Examples:

```
CHVAR:=MEM[$8000];
INTVAR:=ORD(MEM[10]);
MEM[10]:=CHVAR;
MEM[$8000]:=CHR(INTVAR);
```

SHL and SHR: These reserved words may appear as a multiplying operator between two factors in a term of an expression. An integer factor to the left of 'SHL' will be shifted left by the number of bits represented by the value of the integer factor to the right of 'SHL'. Similarly, 'SHR' will cause a shift right.

The following example shows how to store an address value in memory:

```
(* store low half *)
MEM[STOREADDR]:=CHR(ADDRVALUE);
(* store high half *)
MEM[STOREADDR+1]:=CHR(ADDRVALUE SHR 8)
```

PREDECLARED IDENTIFIERS

In addition to the reserved words defined by standard Pascal, there are words which are predeclared by standard Pascal. They differ from reserved words in that you may redefine them for your own use. If you do so, then your definition of these standard identifiers will replace the defined usage in Pascal. However, doing so is not recommended.

All three standard constant identifiers have been implemented in KMMM Pascal. They are 'FALSE', 'TRUE' and 'MAXINT'. Since a variable of type 'INTEGER' holds 15 bits plus a sign bit in KMMM Pascal, the predeclared value of 'MAXINT' is 32767.

All four standard type identifiers have been implemented in KMMM Pascal. They are 'BOOLEAN', 'CHAR', 'REAL' and 'TEXT'. A variable of type 'REAL' in KMMM Pascal occupies five bytes in memory and has the same format as a floating point variable in BASIC.

KMMM Pascal has also implemented a type called 'STRING', which is a standard type identifier in UCSD Pascal. Whenever the word 'STRING' is used, it may be followed by a numeric literal enclosed in square brackets, which specifies the maximum size of the 'STRING'. If size is not specified, the size defaults to 80 characters. The limit of the size of a 'STRING' is 196 characters.

Examples of valid declarations:

```
    VAR FIRSTNAME: STRING[12];
        LASTNAME: STRING[25];
        ADDRESS1, ADDRESS2: STRING[30];
        CITY: STRING[20];
        STATE: STRING[2];
        ZIP: STRING[5];
        LAZY: STRING;  (* SAME AS STRING[80]; *)
```

Both standard file identifiers have been implemented in KMMM Pascal. They are 'INPUT' and 'OUTPUT'.

The remaining standard identifiers are discussed in following sections. The complete list of standard identifiers appears in Appendix D.

SUPPLIED FUNCTIONS

Standard Pascal defines a number of functions. These are 'ABS', 'ARCTAN', 'CHR', 'COS', 'EOF', 'EOLN', 'EXP', 'LN', 'ODD', 'ORD', 'PRED', 'ROUND', 'SIN', 'SQR', 'SQRT', 'SUCC' and 'TRUNC'. KMMM Pascal has now implemented all of these functions, according to the standard.

In addition, KMMM Pascal supplies additonal predeclared functions. Some of these functions are equivalent to functions supplied by UCSD Pascal. The additional functions in KMMM Pascal are:

ANDB: Return value is of type 'INTEGER' and is the Boolean 'AND' of two arguments of type 'INTEGER'.

```
ANDB(15,3) returns 3
```

CONCAT: Return value is of type 'STRING' and is the concatenation of two or more arguments of type 'STRING'. This function is equivalent to the function of the same name in UCSD Pascal. KMMM Pascal, in addition, allows any of the arguments to be of type 'CHAR' and will automatically generate the necessary type conversion logic.

```
CONCAT('ABC','D') returns 'ABCD'
CONCAT('AB','C','D','EF') returns 'ABCDEF')
```

COPY: Return value is of type 'STRING'. This function is equivalent to the function of the same name in UCSD Pascal and to the 'MID$' function in BASIC.

```
COPY('ABCD',1,1) returns 'A'
COPY('ABCD',2,1) returns 'B'
```

DELETE: Return value is of type 'STRING'. This function is the inverse of the 'COPY' or 'MIDSTR' function. There is a procedure named 'DELETE' in UCSD Pascal, but it is implemented as a function in KMMM Pascal, due to an error in a text used as a reference.

```
DELETE('ABCD',1,1) returns 'BCD'
DELETE('ABCD',2,1) returns 'CD'
```

GETKEY: Return value is of type 'CHAR' and is the value of a keystroke on the system keyboard. If no key is pressed, then this function returns the value CHR(0). There are no arguments for this function.

INKEY: Return value is of type 'CHAR' and is the value of a keystroke on the system keyboard. If no key is pressed, then this function will wait until a key is pressed. There are no arguments for this function.

LEFTSTR: Return value is of type 'STRING'. This function is equivalent to the 'LEFT$' function in BASIC.

```
LEFTSTR('ABCD',1) is 'A'
LEFTSTR('ABCD',2) is 'AB'
```

LENGTH: Return value is of type 'INTEGER' and is the length of the argument, which is of type 'STRING'. This function is equivalent to the function of the same name in UCSD Pascal.

```
LENGTH('ABCD') is 4
```

MIDSTR: Synonym for the function named 'COPY'.

NOTB: Return value is of type 'INTEGER' and is the Boolean 'NOT' of a argument of type 'INTEGER'.

```
NOTB(15) returns -16
```

ORB: Return value is of type 'INTEGER' and is the Boolean 'OR' of two arguments of type 'INTEGER'.

```
ORB(5,3) returns 7
```

POS: Return value is of type 'INTEGER', which represents the position of the first argument within the second argument. Both arguments must be of type 'STRING'. If the first argument occurs several times within the second argument, the 'POS' function will return only the first occurance. If the first argument does not occur within the second argument, the 'POS' function will return a zero value. The 'POS' function will accept an expression of type 'CHAR' as the first argument, but not as the second argument. This function is equivalent to the function of the same name in UCSD Pascal.

```
POS('A','ABCD') is 1
POS('B','ABCD') is 2
POS('C','ABCABC') is 3
POS('E','ABCD') is 0
```

RIGHTSTR: Return value is of type 'STRING'. This function is equivalent to the 'RIGHT$' function in BASIC.

```
RIGHTSTR('ABCD',1) is 'D'
RIGHTSTR('ABCD',2) is 'CD'
```

RND: Return value is of type 'REAL' and is a random value. This function requires an argument of type 'INTEGER' or 'REAL'. If the argument is a positive value, the function will return a new random value. If the argument is a negaive value, the function will first alter the internal seed value before returning a new random value.

SUPPLIED PROCEDURES

Standard Pascal defines a number of predeclared procedures. These are 'GET', 'DISPOSE', 'NEW', 'PACK', 'PAGE', 'PUT', 'READ', 'READLN', 'RESET', 'REWRITE', 'UNPACK', 'WRITE' and 'WRITELN'. Of these, KMMM Pascal has not implemented the 'DISPOSE', 'PACK', 'PAGE' and 'UNPACK' procedures. All of the implemented standard procedures, except 'NEW', are discussed in a following section.

The 'NEW' procedure is used to allocate dynamic storage. It may have only one argument, which is a variable identifier whose base type is 'RECORD'.

In addition, KMMM Pascal supplies additonal predeclared procedures. Some of these procedures are equivalent to procedures supplied by UCSD Pascal. The additional procedures in KMMM Pascal are:

CLOSE: The 'CLOSE' procedure may be used to close a file opened via a 'RESET' or 'REWRITE' procedure. The 'CLOSE' procedure requires a single argument which is the identifier of the file to be closed. Use of this procedure is not required, since the run time package closes all open files when a KMMM Pascal machine language program ends. However, it must be used if you intend to access files on different diskettes inserted in the same disk drive during the course of a run.

ERASE: The 'ERASE' procedure may be used to erase or scratch a file from a diskette. The 'ERASE' procedure requires a single argument of type 'STRING', which is the name of a file to be deleted from a diskette. The argument must include a drive specifier.

```
ERASE('O:DISKFILE');
```

EXIT: This procedure is similar to the procedure of the same name in UCSD Pascal. The 'EXIT' procedure may be used to exit from a procedure or function. The 'EXIT' procedure requires a single argument which is the identifier of the procedure or function in whose body the 'EXIT' is located. To exit from the main body of the program, use the identifier 'PROGRAM'.

RENAME: The 'RENAME' procedure may be used to change the name of a file on a diskette. The 'RENAME' procedure requires two arguments of type 'STRING'. The first argument is the name of an existing file on diskette. The second argument is the new name to be given to the file. The second argument must have a drive specifier, however, the specifier is optional in the first argument.

```
RENAME('OLDNAME','0:NEWNAME');
```

LIMITATIONS

The following limits exist on declarations:

1. Limit of 32 different enumerated types

2. Limit of 32 different ARRAY types

3. Limit of 32 different RECORD types

4. The maximum size of a string used as a parameter for a function or a procedure is 80 bytes

5. A structured type used as a parameter for a function or a procedure must be declared as a variable parameter

6. The parameter list in the heading of a function or procedure declaration may contain a maximum of two STRING types

7. The maximum size of the component type of an ARRAY is 255 bytes

8. The total number of bytes allocated to the fields comprising a RECORD type may not exceed 255

9. The fields comprising a RECORD type may not be structured

INPUT/OUTPUT PROCEDURES

The procedures 'READ' and 'READLN' allow the user to read data from the keyboard, a cassette file or a disk file into a variable or a list of variables. The procedure 'GET' allows the user to fill the file buffer from those devices. The procedures 'WRITE' and 'WRITELN' allow the user to write an expression or list of expressions to the console screen, a cassette file or a disk file. The procedure 'PUT' allows the user to empty the file buffer to those devices. The procedures 'RESET' and 'REWRITE' must be executed in order for data transfer to take place between the program and cassette or disk files.

The 'READ(LN)' and 'WRITE(LN)' procedures may only be used with file identifiers whose component type is 'CHAR'. The 'GET' and 'PUT' procedures may only be used with file identifiers whose component type is 'RECORD'.

The following program fragment gives some examples of use of the READ procedure:

```
VAR CHVAR: CHAR;
    INTVAR: INTEGER;
    REALVAR: REAL;
BEGIN
  READ(CHVAR);   (* Example 1 *)
  READ(INTVAR);  (* Example 2 *)
  READ(REALVAR); (* Example 3 *)
  READ(INTVAR$); (* Example 4 *)
END.
```

Example 1 will cause a single character to be read from the keyboard and stored in the variable CHVAR. Examples 2, 3 and 4 will cause a combination of character input and data conversion. The number of characters read depends on the declared type of the variable being read into. For 'INTEGER' and 'REAL' variables, characters will be read until a stopping character is encountered. For 'INTEGER' variables, the stopping character is any character other than the digits '0' to '9'. For 'REAL' variables, the stopping character is any character other than the digits '0' to '9', the decimal point or the letter 'E'. For hexadecimal input into an 'INTEGER' variable, the stopping character is any character other than the digits '0' to '9' and the letters 'A' to 'F'.

This method of input is quite different from the way BASIC operates. In BASIC, entering the string '39X' in response to a request for numeric input will result in '?REDO FROM START'. In KMMM Pascal, entering the string '39X' in response to a request for numeric input will not result in an error message. A display of the variable read into will show a value of 39.

The following program fragment gives some examples of use of the WRITE procedure:

```
CONST CHCONST = 'A';
      STRCONST = 'XYZ';
      INTCONST = 13;
      REALCONST = 2.3E+4;
BEGIN
  WRITE(CHCONST);
  WRITE(INTCONST-2,STRCONST);
  WRITE(1=1,REALCONST);
  WRITE(INTCONST$)
END.
```

The output of this program is:

```
A11XYZTRUE23000000D
```

If all the WRITE's are changed to WRITELN, then the output is:

```
A
11XYZ
TRUE23000
000D
```

If the first or only parameter of a 'GET', 'READ', 'READLN', 'PUT', 'WRITE' or 'WRITELN' procedure is file identifier, then the I/O operation will involve an external file instead of the system console. Before attempting a 'READ(LN)' or 'WRITE(LN)' to an external file, instead of the console, a 'RESET' or 'REWRITE' procedure must be executed to establish a data path for that file. The 'RESET' procedure is used for input files and the 'REWRITE' procedure for output files. The first parameter of either a 'RESET' or 'REWRITE' procedure must be a file identifier. The second parameter is the external file name and must be a string expression. The following program fragment shows three different methods for opening the same disk file on drive 0 for input:

```
CONST CONSTNAME = '0:ABC';
VAR INFILE : TEXT;
    VARNAME : STRING[16];
    .................
  RESET(INFILE,'0:ABC');
  ......................
  RESET(INFILE,CONSTNAME);
  ........................
  VARNAME := '0:ABC';
  RESET(INFILE,VARNAME)
```

If the file 'ABC' does not exist on drive 0, then the function 'EOF(INFILE)' will return the value 'TRUE' immediately after the reset procedure is completed. Older versions of the run time package would abort the program if a specified file did not exist.

A more complete discussion of external file access appears in the following section of this manual.

EXTERNAL FILE ACCESS

Normally, a 'GET', 'READ' or 'READLN' procedure will read from the console keyboard and a 'PUT', 'WRITE' or 'WRITELN' procedure will write to the console screen. However, input or output may be directed to an external file if the first parameter of a 'READ', 'READLN', 'WRITE' or 'WRITELN' is a variable defined as being of type 'TEXT'. To do so, a data path must first be established. The 'RESET' procedure must be executed for a file that you intend to 'READ' or 'READLN' from; the 'REWRITE' procedure must be executed for a file that you intend to 'WRITE' or 'WRITELN' to. A file opened with a 'REWRITE' may also be read from, but onlv if appropriate for the device being used.

The first parameter of a 'RESET' or 'REWRITE' procedure must be a file identifier. This first parameter is followed by a variable number of parameters. If the next parameter is an expression of type 'STRING', the Compiler considers it to be the file name. This file name expression must be the last parameter. This two parameter format will be referred to as the 'automatic' format and can only be used with sequential files on cassette or diskette. If a diskette file is indicated, device 8 will be used.

If the second parameter is not an expression of type 'STRING', then it must be an expression of type 'INTEGER'. The run time package will use this value as the IEEE device number. The second parameter must be followed by a third parameter, which must be an expression of type 'INTEGER' and will be passed to the IEEE device as the 'secondary address'. The fourth parameter must be an expression of type 'STRING', but may be omitted if a file name is not appropriate for the IEEE device being used. This three or four parameter format will be referred to as the 'manual' format.

The 'automatic' format is so designated because it will use the command channel to check for a successful file open if the file name prefix indicates a disk file. It will also add an ',S,R' or ',S,W' suffix to the file name if the file name prefix indicates a disk file. For those of you who may wish to read or write 'PGM' files, then a ',P' suffix is allowable in the file name and will cause the ',S,R' or ',S,W' suffix normally appended, to change to ',P,R' or ',P,W'.

The 'manual' format will do nothing extra for you. It will not adjust the file name; it will not check for a successful file open. If the device being used, such as the standard CBM disk drives, requires special characters in the file name, you must provide them. Any special command channel programming must be explicitly coded.

The standard distribution diskette now contains additional sample programs that show how to access files. The program 'PCCEXMP' shows how to read from and write to the CBM disk command channel in order to scratch a file on a CBM diskette. The program 'PLSTCBMD' shows how to access the directory of a CBM diskette as a file. The program 'PRANDCR' will create a random file and the program 'PRANDUPD' is a very simple random file update program. Please note that these two random access programs will onlv work with disk drives that have DOS 2.0 or higher.

OUTPUT FIELD FORMATTING

KMMM Pascal supports field width specifications in 'WRITE' or 'WRITELN' procedures. A field width specification consists of a colon followed by an integer expression. The field width specification may be used with any data type. A second specification is possible with the 'REAL' data type, and specifies the number of digits to the right of the decimal point to be displayed.

Integer expressions may have a '$' suffix to indicate that output in hexadecimal is desired. The standard width for hexadecimal output is four character positions. By using a '$:2' specification, the field will be a hexadecimal value two positions wide instead of four.

Example of a valid statement:

```
WRITE(1:2,1.1:4,'?':2,'ABC':4,TRUE:5,16$:2);
```

The output of the above statement would be:

```
' 1 1.1 ? ABC TRUE10'
```

Padding spaces are added to the left of the output field, but only if the size of the output field is less than the width specification. ie Right Justified.

If the width specifications were omitted from the above example, the statement would be:

```
WRITE(1,1.1,'?','ABC',TRUE,16$);
```

The output would then be:

```
'11.1?ABCTRUE0010'.
```

RUN TIME CONSIDERATIONS

A program generated by KMMM Pascal can be handled as if it were a BASIC program text file. The program consists of a single BASIC line followed by machine language. The machine language consists of a run time package plus the machine language generated from your Pascal source file. The run time package is 6K on PET/CBM systems, 8K on the 64.

Space for variables declared in a KMMM Pascal program is allocated immediately after the last byte of machine language generated from your source file. Therefore, the space available for variables depends on the amount of RAM that exists beyond the end of your program.

The first version of level IV of KMMM Pascal was the first to implement dynamic storage allocation. The storage allocated to pointer variables grows down from the top of user memory. The high memory address is taken from the BASIC high memory address in the versions of KMMM Pascal for PET/CBM machines. The high memory address defaults to $C900 in the version of KMMM Pascal for the 64.

To change this default address to a lower value, you must poke two values into the generated program before it is used for the first time. The first address to be changed is 4708. The current content of this address is 201, the decimal equivalent of $C9. The second address to be changed is 4712. The current content of this address is 199, two less than the first address. To change the default address from $C900 to $C000, you would poke 192 into 4708 and 190 into 4712, after verifying that the current contents of these addresses are exactly as stated.

OPERATING INSTRUCTIONS

The KMMM Pascal Compiler is named 'COMPILER-KMMM' on the distribution diskette. You may wish to shorten or change the name for operating convenience, but the following instructions assume that the name of the compiler program starts with 'CO'. Do not change the name of the Translator, as its name is embedded in the Compiler.

There are two ways of loading programs into your Commodore system from disk. The first is to use the BASIC disk support commands that are part of Commodore BASIC. The second involves the use of a special disk support program, sometimes referred to as the DOS Wedge. This special disk support program is discussed in detail in Appendix A.

The standard method of executing the Compiler is to enter the BASIC command 'LOAD "CO*",8'. This should cause the activity light on the disk drive to come on, as the Compiler is loaded into memory. When the loading process is complete, the 'READY.' message should appear. When it does, enter the BASIC command 'RUN'.

Owners of PET/CBM machines with 4.0 BASIC installed in ROM may use the slightly shorter BASIC command 'DLOAD "CO*"' to load the Compiler into memory. This command must be followed by the 'RUN' command to actually start the Compiler. [illegible]

Finally, users who decide to utilize the wedge program may execute the Compiler by entering the up arrow character followed by the letters 'CO*'.

After printing a copyright statement, the Compiler will prompt the user for the name of the KMMM Pascal source file to be compiled. See the Appendix B for rules about specifying file names. Simply pressing the return key or entering a file name with a cassette drive designator ('T:' or 'U:') will cause the Compiler to read the source file from cassette. Entry of a file name without a drive designator will cause the directory (or directories) of the disk drive(s) to be searched. Entry of a file name with a disk drive designator ('0:' or '1:') will cause only the directory of the specified drive to be searched.

Entry of an up arrow character ('↑') may be used to terminate the Compiler.

After successfully opening the KMMM Pascal source file, the Compiler will issue a prompt based on the state of the screen mode. If the screen is in graphics mode, the prompt will be 'ASCII SOURCE FILE? '. Entry of a 'Y' will cause the screen to be shifted to the text mode; entry of any other character will be ignored.

If the screen is in text mode, the prompt will be 'PETSCII SOURCE FILE? '. Entry of a 'Y' will cause the screen to be shifted to the graphics mode; entry of any other character will be ignored.

Please ignore the following paragraph, if you are using the version of KMMM Pascal for the Commodore 64.

PET only

~~If the response to either variant of the above prompt results in the screen being in the text mode, then the~~ Compiler will prompt with 'OLD CHAR. GEN. ROM? '. Enter a 'Y' if you have an old character generator ROM in your PET, otherwise just press the return key.

The Compiler will then prompt with 'USE PRINTER? '. Entry of a 'Y' will direct the source listing to a printer; entry of any other character will cause the source listing to appear on the system console display. If the source listing is directed to the printer, each source line is preceeded by its line number, which can be used when editing the source file.

If you decide to route the source listing to a printer, you will receive three more prompts. The first is 'DEVICE NUMBER? '. Enter the printer device number, which must be a single digit. The second prompt is

'GENERATE LINE FEED? '. Enter a 'Y' if your printer does not automatically generate a line feed when it receives a carriage return character. The third prompt is 'DISPLAY P-CODE NUMBERS? '. Enter a 'Y' if you wish to have the P-code numbers displayed between the source line number and the source line itself. The P-code number is the number of the first P-code generated by the associated line of the source file.

If an error is discovered by the Compiler, an up arrow ('↑') is displayed under the source element causing the error, followed by an error message describing the error. The user must then use the Editor/Compiler or the Editor to create a corrected version of the source file and then re-run the Compiler.

If no errors are found in the source file, the Compiler will then prompt with 'DUMP P-CODE TO DISK? '. Enter a 'Y', if you wish to transfer the P-code stored in memory to a disk file. The file name for this file is created by appending '.PC' to the program identifier in the source file. The program identifier is the identifier that follows the reserved word 'PROGRAM' at the beginning of the source file. If this program identifier is longer than eight characters, only the first eight are used to create the file name for the disk P-code file. Dumping the P-code to disk is of no value at this time, since the P-code disassembler and a version of the Translator that reads P-code stored on disk are not yet ready for general distribution.

The Compiler will then prompt with 'PRESS RETURN WHEN READY: '. When you press the return key, the Compiler will reset the screen mode to what it was at the time you loaded the Compiler.

If there are no errors, then the Compiler will prompt with 'EXECUTE TRANSLATOR? Y'. If you change the default 'Y' to some other character, then the Compiler will prompt with 'COMPILE ANOTHER FILE? '. Entry of a

'Y' will cause the Compiler to restart; entry of any other character will cause a return to BASIC.

Simply pressing the return key in response to the 'EXECUTE TRANSLATOR? Y' prompt will cause the Translator to load into memory and execute. A reminder: if you have only a single disk drive and removed the diskette with the Translator on it in order to insert the source file diskette, be sure to re-insert the Translator diskette before pressing the return key, which will cause an attempt to load the Translator into memory.

After printing a copyright statement, the Translator will display a message as it begins each of its internal phases and will display an asterisk for each 10 P-codes it processes. This will let those of you who compile large programs know that the Translator is alive and well, and has not disappeared off into limbo.

In a first pass, the Translator counts the number of P-codes referred to by other P-codes. After completion of this first pass, it displays a message 'n LABELS' on the system console, where 'n' is the count of P-codes referred to.

In a second pass through the generated P-code, the Translator generates machine language. At the end of the second pass, it displays run statistics on the system console and returns to BASIC.

The Translator generates a machine language program that consists of a 6K (8K in the C64 version) package of support subroutines followed by the machine language generated from your source file. This combined block of machine language is preceeded by a BASIC 'SYS' statement which allows the program to be executed by entering a BASIC 'RUN' command.

The Translator sets the program boundary pointers used by Commodore BASIC so that a standard BASIC 'SAVE' or 'DSAVE' command can be used to store the entire generated program on cassette or diskette.

Because of this, it is possible to defer the step of saving the generated program until after the program is tested to see if it runs as desired.

ERROR MESSAGES

Most of the error messages displayed by the Compiler or the Editor/Compiler are self-explanatory. There is one message, however, which is not. That one is 'END TEXT FOUND'. This message will appear if you have a '(*' without a matching '*)'. Sometimes the onlv way to locate this problem is to direct the output of the Compiler to the printer. If the P-code numbers get stuck on one value, then the problem is in the area of the last increment in the P-code number.

The run time package will disnlay an error and terminate the program if certain run time errors occur. These are:

1. 'n FILE - BAD FILE NAME'
2. 'n FILE NOT OPEN'
3. 'n FILE READ ONLY'
4. 'n FILE - DISK ERROR nn'
5. 'DIVIDE BY ZERO'
6. 'OUT OF MEMORY'

The 'n' in the first four messages is the sequential number assigned by the Compiler as it encounters each file identifier in a variable declaration section of a source file.

Message number one occurs whenever the run time package finds a file name that violates the file name syntax rules laid out in Appendix B. The most common mistake made by users is to use the letter 'O', instead of the digit '0' in the drive specifier.

Message number two occurs if you attempt to perform an I/O operation to an external file without first doing either a 'RESET' or 'REWRITE' procedure for that file identifier.

Message number three occurs if you attempt to perform an output operation to an external file that was opened using a 'RESET' procedure.

Message number four should be self-explanatory. The 'nn' is the Commodore disk error code sent by the DOS in the disk drive. Refer to the error codes in your Commodore disk drive manual.

Message number five occurs if you attempt to divide by zero.

Message number six occurs if the stack pointer address and the bottom of the 'heap' get within 512 bytes of each other. The stack grows upwards in memory and the storage space for records allocated via the 'NEW' procedure (the 'heap') grows downward.

DISKETTE COPY

The fastest and easiest way to make a working copy of the distribution diskette is to own a dual drive disk unit, such as the Commodore 4040 or 8050 or the Micro Systems Devices SD-2. These drives have disk copy logic built into them.

If you do not own a dual drive disk unit, you can transfer all the 'PRG' type files (except 'CBMWEDGE', 'CIESUPP' and 'BOTHSUPP') from the distribution diskette to the working copy using the standard BASIC verbs 'LOAD' and 'SAVE'. The remaining files must be transferred using some kind of copy program.

There are a large number of diskette backup programs available. The best of these are available from commercial sources, although there are some reasonably good public domain programs available. Check with your local Commodore user group.

If you do not own any kind of copy program. there is a Pascal source file on the distribution diskette, named 'PCOPYPRG' that, after being compiled, can copy the modules that can not be transferred using the 'LOAD' and 'SAVE' BASIC commands. These are: 'CBMWEDGE', 'CIESUPP' and 'BOTHSUPP'.

The 'SEQ' type files may be transferred using the Editor. Use the command sequence 'GRfilename!Y!GC!!' to read a specific 'SEQ' file in from the distribution diskette, where 'filename' is the name of the file to be transferred. Then, remove the distribution diskette, insert the new working copy and issue the command sequence 'GWO:filename!P!GC!!'. Both of the above command sequences assume you have elected to use '!' as the escape character.

FOREIGN DEVICES

Foreign devices are those hardware accessories manufactured by companies other than Commodore. Usually, these devices cause no problems for KMMM Pascal. However, some devices require that certain memory locations be left undisturbed.

We provide re-written support routines so that the 'CIE', an IEEE adaptor from Micro Systems Devices, can be used with KMMM Pascal. See appendix A for instructions on using these special routines.

We generated a variation of version IV.4 of KMMM Pascal that worked with the 'Screenmaker', an 80 column video board from CGRS Microtech. However, that version is now obsolete. If you need a version of KMMM Pascal that works with 'Screenmaker', please contact Wilserv Industries.

We have spent quite a few hours analyzing the code that accompanies the C64-link from Richvale Communications of Canada, an IEEE adaptor. We still have not figured out how to resolve the memory conflict between this device and KMMM Pascal. Some users have had success doing so. We would like to hear from those users.

We believe it would not take much to make the 80 column video display card from Data-20 work with KMMM Pascal. Until we get our hands on one, we won't know for sure.

REPORTING PROBLEMS

We have exerted a great deal of effort to insure that the programs that comprise KMMM Pascal run without error. However, due to the nature of a compiling system, you may find an error in the system. We therefore urge you to report problems. If you don't report problems, we can't fix them. We are especially interested in receiving examples of misleading or unclear error messages.

Sometimes, if an error is trivial, and you find a way around the problem quickly, there is a tendency to forget it happened. Therefore, if you find a problem in a source file, we ask that you take a moment to copy the source file to a spare diskette and send that diskette along with a brief note describing the problem to us, at the address on the front of this upgrade manual.

Furthermore, because of the current heavy work load at Wilserv Industries, we will very likely insist that you send in the source file that shows the error you have discovered. Last year, when the first versions of level IV were released, there were numerous problems and some were so major, they were easily described over the phone. However, at the current level of maturity, we will almost always need your source file to locate the problem.

USER SUPPORT

Our phone number is (609)227-8696. This number is valid from 10 A. M. to 10 P. M. (EST or EDT), Monday to Friday. Saturday, the hours are 10 to 2. No calls accepted on Sunday.

If you have questions or a problem, please feel free to call. However, we do ask that you carefully read all the enclosed documentation. It is not the easiest material to read, but we believe that all the information needed to use the package has been presented.

Before calling, make sure you have followed the instructions in the section of this manual titled 'REGISTRATION FORM'. Also, be aware that if the above number is answered on the first ring, it is an answering machine that is answering. This answering machine is controlled by a timer and will only be on from noon to five P. M. (EST or EDT), Monday to Friday. If you call around 5 P. M. and get the answering machine, you may be cut off by the timer.

Occasionally, the answering machine will be on from 10 A. M. to noon. This occurs only when we close the office, either due to a minor emergency or because Wilserv is temporarily closed for vacation.

If you get frustrated trying to reach us by phone, please complain in writing to our mailing address.

LOADING PROGRAMS

The standard BASIC command to load a program from a diskette into memory is 'LOAD "filename",8'. Users of PET/CBM machines with 4.0 BASIC installed may use the slightly more convenient command 'DLOAD "filename'. In either case, the BASIC command 'RUN' must be entered to actually start the program after the load operation is complete.

An alternate method of running a program stored on diskette is to use the disk support program known as the 'DOS WEDGE'. With this program in place, a program may be both loaded into memory and started by entering the command '↑filename'.

The standard 'WEDGE' program for PET/CBM machines will work with KMMM Pascal, therefore the distribution diskette for these systems does NOT contain a 'WEDGE' program.

A special version of the 'WEDGE' IS supplied as part of the version of KMMM Pascal for the Commodore 64.

The first file on the distribution diskette, when loaded into memory and run, will cause one of three disk support routines to be loaded into memory. All three function identically, but the one named 'CBMWEDGE' is for use with the Commodore 1541 serial disk drive, the one named 'CIESUPP' is for use with the IEEE adaptor from Micro Systems Devices (CIE) and the one named 'BOTHSUPP' is for users who have a 1541, but use the CIE adaptor to drive an IEEE printer.

To install the proper disk support routine, simply enter

'LOAD "BOOT",8'. If you are using the CIE adaptor with an IEEE disk unit, you must first enter 'SYS57278' to allow the 'LOAD' to function. If you are using the CIE adaptor with a 1541 disk unit, do NOT use the 'SYS57278' at this point. After the 'LOAD' is complete, enter 'RUN'. Respond with a 'Y' to the 'CIE IEEE ADAPTOR? ' prompt only if you have a 1541, but wish to use the CIE for an IEEE printer.

Once the disk support routine is activated, the component programs of KMMM Pascal can be loaded into memory and run by simply entering the up arrow character followed by the program name. It is not necessary to enclose the program name in quotes.

A display of the directory of a diskette can be obtained by entering '@$'.

CIE users may enter '@x', where x is the British pound symbol key, as a substitute for 'SYS57278', after the 'CIESUPP' has been installed.

FILE NAME SYNTAX

If a null file name is entered, the system will default to cassette drive 1. If opening an input file, the first file found will become the file read. If opening an output file, the file will be written without a name.

A unit designator must be part of the file name when opening either a cassette input file, a cassette output file or a disk output file. A unit designator consists of a single character followed by a colon. For a disk file, the single character is either '0' or '1'; for a cassette file, the single character is either 'T' (first cassette drive) or 'U' (second casette drive).

If a unit designator does not preceed the file name when opening an input file, the system will assume that a disk drive is to be used and search the directories of both drives for the specified file name. A unit designator (either '0:' or '1:') may be used to limit the directory search to a siingle drive. The use of the unit designators 'T:' (first cassette drive) or 'U:' (second drive) when opening an input file will cause a file to be read from either cassette drive. If the file name consists of only the unit designator, the first file found will become the file read.

When opening an output file, the file name must begin with a unit designator. Since a cassette file may be created without a name, a unit designator alone ('T:' or 'U:') is sufficient for a cassette output file name. The disk unit designators ('0:' or '1:') must be followed by at least one character.

RESERVED WORDS

STANDARD PASCAL

AND	END	NIL	SET *
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO *	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH *
ELSE	MOD	REPEAT	

KMMM PASCAL

CALL	MEM	SHL	SHR

* Not implemented in current version of KMMM Pascal

PRE-DEFINED IDENTIFIERS

STANDARD PASCAL

CONSTANTS:

FALSE	TRUE	MAXINT	

TYPES:

BOOLEAN	INTEGER	REAL	TEXT
CHAR			

FUNCTIONS:

ABS	EOLN	PRED	SUCC
ARCTAN	EXP	ROUND	TRUNC
CHR	LN	SIN	
COS	ODD	SQR	
EOF	ORD	SQRT	

PROCEDURES:

GET	PAGE *	READLN	UNPACK *
NEW	PUT	RESET	WRITE
PACK *	READ	REWRITE	WRITELN

FILES:

INPUT	OUTPUT		

* Not implemented in current version of KMMM Pascal

KMMM PASCAL

TYPE:

STRING			

FUNCTIONS:

ANDB	GETKEY	MIDSTR	RIGHTSTR
CONCAT	INKEY	NOTB	RND
COPY	LEFTSTR	ORB	
DELETE	LENGTH	POS	

PROCEDURES:

CLOSE	ERASE	EXIT	RENAME

BIBLIOGRAPHY

Cooper, Doug, STANDARD PASCAL USER REFERENCE MANUAL, W. W. Norton, 1983

Grogono, Peter, PROGRAMMING IN PASCAL, Addison-Wesley, 1978

Jensen, Kathleen and Wirth, Niklaus, PASCAL USER MANUAL AND REPORT, Springer-Verlag, New York, 1974

Kusche, Willi, ZOOM PASCAL, Abacus Software, Grand Rapids, 1983

Ledgard, H., Hueras, J. and Nagin, P., PASCAL WITH STYLE, Hayden, 1979

Welsh, Jim and Elder, John, INTRODUCTION TO PASCAL, Prentice-Hall, 1979

Wirth, Niklaus, ALGORITHMS + DATA STRUCTURES = PROGRAMS, Prentice-Hall, 1976

Yuen, Herbert and Chung, Kin-Man, "A Tiny Pascal Compiler", BYTE Magazine, September-November 1978

Zaks, Rodnay, AN INTRODUCTION TO PASCAL, Zybex, 1980

THE JOURNAL OF STRUCTURED LANGUAGES, West Publishing (periodical)