M20 PERSONAL COMPUTER

PASCAL Language

User Guide

olivetti

PREFACE

This manual is produced for programmers using the M20 to create PASCAL language programs. The PASCAL language available on the M20 is the MS-PASCAL developed by "Microsoft Corporation", and which is described in the "M20 PASCAL Language Reference Manual". The reference manual gives the complete instruction set and explains other fundamental principles of PASCAL programming. It is assumed that the reader of this User Guide has a working knowledge of both The PASCAL language and PCOS. This User Guide explains how to use the PASCAL compiler and the linker to produce executable program files for PCOS release 3.0, and supplies further information concerning this particular PASCAL implementation.

This User Guide is divided in two parts:

Part I - describes how to create an executable program file starting from a PASCAL language source file.

Part II - details the functions and procedures (contained in two library files) available on the M20.

REFERENCES:

PASCAL Language Reference Manual
(Code 3987710 Q (0))

PCOS (Professional Computer Operating System) User Guide
(Code 3985280 D (0))

I/O with External Peripherals User Guide
(Code 3982300 N (2))

Assembler User Guide
(Code 3987670 L (1))

M20 Hardware Architecture and Functions
(Code 4100630 W (0))

DISTRIBUTION: General (G)

EDITION: July 1983

RELEASE: 3.0

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

PREFACE

This manual is produced for programmers using the M20 to create PASCAL language programs. The PASCAL language available on the M20 is the MS-PASCAL developed by "Microsoft Corporation", and which is described in the "M20 PASCAL Language Reference Manual". The reference manual gives the complete instruction set and explains other fundamental principles of PASCAL programming. It is assumed that the reader of this User Guide has a working knowledge of both The PASCAL language and PCOS. This User Guide explains how to use the PASCAL compiler and the linker to produce executable program files for PCOS release 3.0, and supplies further information concerning this particular PASCAL implementation.

This User Guide is divided in two parts:

Part I - describes how to create an executable program file starting from a PASCAL language source file.

Part II - details the functions and procedures (contained in two library files) available on the M20.

REFERENCES:

PASCAL Language Reference Manual
(Code 3987710 Q (0))

PCOS (Professional Computer Operating System) User Guide
(Code 3985280 D (0))

I/O with External Peripherals User Guide
(Code 3982300 N (2))

Assembler User Guide
(Code 3987670 L (1))

M20 Hardware Architecture and Functions
(Code 4100630 W (0))

DISTRIBUTION: General (G)

EDITION: July 1983

RELEASE: 3.0

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

CONTENTS

1. INTRODUCTION

ABOUT THIS CHAPTER

This chapter is a brief introduction to the M20 PASCAL package. This chapter also provides an overview of this user guide.

CONTENTS

PROGRAMMING IN PASCAL ON THE M20

The M20 PASCAL language is a compiled language. This means that a PASCAL language program must be written in an Editor environment; on the M20 this can be done in the Video File Editor environment which is described in the "M20 PCOS (Professional Computer Operating System) User Guide". This edited version of the program containing PASCAL statements is known as the source file. The structure of the source file and the PASCAL language as implemented on the M20 are detailed in the "M20 PASCAL LANGUAGE Reference Manual". This user guide details all the subsequent steps required before the program can be successfully executed.

The next step is to compile the program using the PASCAL compiler. The PASCAL compiler takes a source file in input and creates a new file called the object file.

The final step in creating an executable load file is to link object files using the LINK command. LINK takes one or more object files in input and creates an executable binary load file. Note that object files created using other computer languages can be LINKed to object files output by the PASCAL compiler.

THE M20 PASCAL PACKAGE

The M20 PASCAL package contains the PASCAL compiler, the LINK command, and the Video File Editor mentioned above. Also included in the package is the MLIB command for creating library files of object modules. The package also includes other data files required by the compiler and library files of object modules which implement real number arithmetic and for interfacing some of the PCOS system calls and the routines of the M20 graphics package.

SYSTEM REQUIREMENTS

The Olivetti PASCAL compiler will run on any M20 with a minimum of 384K bytes of random access memory. For easier operation, an M20 with two floppy disk drives or one floppy disk drive and a hard disk is recommended.

ABOUT THIS MANUAL

Part one of this manual (chapters 1-5) describes the operation of the M20 PASCAL Compiler and the M20 linker from the most rudimentary procedures to more advanced topics that may be of interest only to experienced programmers. It is assumed however that the reader has a working knowledge of both the PASCAL language and PCOS.

Chapter two "Program Development" describes the process of program development. This chapter includes a sample session which provides a step-by-step description of all the stages that follow the writing of a program. This chapter is aimed primarily at users who are new to programming in PASCAL.

Chapters 3 and 4 are complete descriptions of the compiler and the linker respectively. These two chapters are self contained and experienced programmers may want read these chapters right from the start.

Chapter 5 discusses the use of library files and describes the MLIB command for creating library files.

Part two is dedicated to the functions and procedures contained in the two library files available in the M20 PASCAL package; the M20 PASCAL Library and the M20 PASCAL Graphics Library.

The M20 PASCAL Library contains the functions (and one procedure) which interface some of the PCOS system calls. The M20 PASCAL Library is introduced in chapter 6 and the functions (and one procedure) are described in chapter 7.

The M20 PASCAL Graphics Library contains functions and procedures which interface the routines of the M20 Graphics package. The M20 PASCAL Graphics Library is introduced in chapter 8 and the functions and procedures are detailed in chapter 9.

NOTATION CONVENTION

The notation used in this manual to describe command syntax is that defined in the "PCOS User Guide". Refer to the "PCOS User Guide" also for rules governing file names and file identifiers in general.

2. PROGRAM DEVELOPMENT

ABOUT THIS CHAPTER

This chapter provides a short introduction to program development, a multistep process which includes first writing the program, and then compiling, linking, and executing it. For a brief explanation of terms that may be unfamiliar, see Appendix H "Vocabulary."

CONTENTS

CREATING AN EXECUTABLE FILE

A microprocessor can execute only its own machine instructions; it cannot execute source program statements directly. Therefore, before a PASCAL program can be executed, some type of translation, from the statements in the source file to the machine language of the microprocessor, must occur.

Compilers and interpreters are two types of programs that perform this translation. Depending on the programming language in use, either or both types of translation may be available. The M20 PASCAL is a compiled language.

A compiler translates a source program and creates a new file called an object file. The object file contains relocatable machine code that can be placed and run at different absolute locations in memory.

Compilation also associates memory addresses with variables and with the targets of GOTO statements, so that lists of variables or of labels do not have to be searched during execution of your program.

Many compilers, including the M20 PASCAL Compiler, are what are called "optimizing" compilers. During optimization, the compiler reorders expressions and eliminates common subexpressions, either to increase speed of execution or to decrease program size. These factors combine to measurably increase the execution speed of your program.

The M20 PASCAL Compiler has a three-part structure. The first two parts, pass one and pass two, together carry out the optimization and create the object code. Pass three is an optional step that creates an object code listing. Compiling is described in more detail in Chapter 3 "The PASCAL Compiler".

Before a successfully compiled program can be executed, it must be linked. Linking is the process which computes absolute offset addresses for routines and variables in relocatable object modules and resolves all external references by searching through the runtime library (and any other specified library files). On the M20 this operation is done by the LINK command. LINK saves your program on disk as an executable file, ready to run.

It is possible, at link time, to link more than one object module, as well as routines written in Assembly Language or other high-level languages and routines in other libraries. The LINK command is described in greater detail in Chapter 4 "The LINK Command".

PROGRAM DEVELOPMENT

The entire process of program development is outlined below. The process is illustrated schematically in figure 2-1

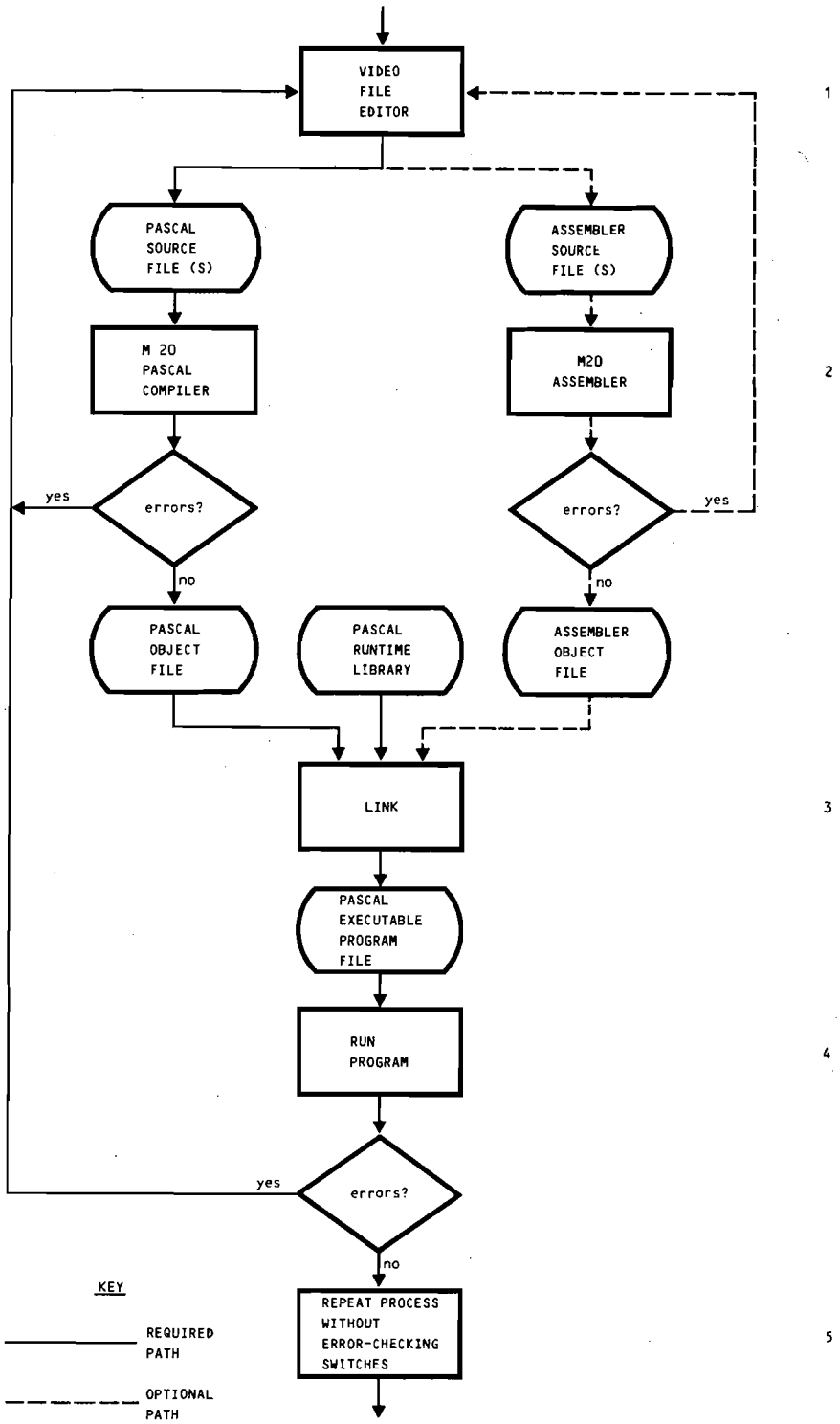


Fig. 2-1 Program Development

1. Create and edit the PASCAL (and M20 Assembler) source file(s).

   Program development begins when you write a PASCAL program using the Video File Editor. Use the text editor also to write any Assembly Language routines you may plan to include.

2. Compile the program with the "/D" ($DEBUG+) switch. Assemble the Assembler source, if any, using the M20 Assembler (ASM) command.

   Once you have written a program, compile it with the M20 PASCAL Compiler. The Compiler flags all syntax and logic errors as it reads your source file. The "/D" Compiler switch (or the corresponding "$DEBUG+" metacommand in the source file) turns on all the error-checking switches. These switches are equivalent to their corresponding metacommands and generate diagnostic calls for all run-time errors(see chapter 3 for more detail on Compiler switches). If compilation is successful, the compiler will create a relocatable object file.

   If you have written your own Assembly Language routines (for example, to increase the speed of execution of a particular algorithm), assemble those routines with the Assembler (ASM) command. The ASM command is described in the "M20 ASSEMBLER User Guide".

3. Link the compiled (and assembled) object files with the runtime library.

   An object file output by the PASCAL compiler must be linked with the runtime library "runtime.lib". LINK will create an executable program file.

4. Execute the program.

   Having compiled the program with $DEBUG+ will cause the executable program to go through extensive (and time consuming) error checking procedures.

   Repeat this process until your program has successfully compiled, linked, and run without errors.

5. Recompile, relink, and rerun with $DEBUG-.

   Repeating the process without the runtime error-checking switches will reduce the amount of time and space required to execute the program.

ERRORS AND WARNINGS

PASCAL errors and warnings can be returned either by the compiler (at compile time) or during program execution (run time). The codes of errors and warnings that can be returned are detailed in Appendix H of the Reference Manual.

DUMPING FACILITIES

Throughout the process of program development the programmer may need to display source files, object files, etc.. This can be done using the PCOS command FLIST, which allows a number of optional features for dumping various types of files. The FLIST command is detailed in the "M20 PCOS User Guide".

SAMPLE SESSION

This sample session provides step-by-step instructions for compiling and linking an M20 PASCAL program.

Creating an executable M20 PASCAL program involves the following steps:

1. Write and save a PASCAL source file.

2. Compile your program with the M20 PASCAL compiler.

   a) Start pass one and enter your filenames in response to the prompts.

   b) Run pass two of the compiler.

   c) Run pass three of the compiler. (This step is optional.)

3. Link your object file to the runtime library "runtime.lib".

4. Execute (i.e., run) your program.

Compiler passes one and two are required. You need to run pass three only if need or want an object listing, which we do for this session.

This sample session makes the following assumptions:

1. You have an M20 with a minimum of 384K bytes of random access memory and two disk drives (0: and 1:).

2. The sample program is already debugged, so that it will compile, link, and execute successfully.

3. An object listing is required, therefore all three passes of the compiler will be run.

4. No compiler switches will be used.

5. There are no problems with data, code, or memory limits.

CREATING AN M20 PASCAL SOURCE PROGRAM

Turn on your computer and load PCOS. You can create a PASCAL program using the Vide File Editor. A PASCAL source file should, in most cases, have the ".pas" extension. For this sample session, we shall use a program called "sort.pas". sort.pas will reside on drive 1.

COMPILING YOUR M20 PASCAL PROGRAM

As mentioned previously, compiling a program is either a two or three-step process, depending on whether or not you choose to produce an object code listing. For the sample session, we will run all three passes.

Pass One

Insert the disk containing the files "pas1.cmd" and "PASKEY" in drive 0. In response to the PCOS prompt, enter

```
pas1
```

This command starts pass one of the M20 PASCAL compiler.

The compiler prints a header then prompts you with the following message:

```
Ready diskette in default drive 0
or enter new default drive number:
```

To this message respond by entering the number 1 followed by a carriage return.

The compiler then prompts you for four filenames:

1. The source filename

2. The object filename

3. The source listing filename

4. The object listing filename

Respond to the prompts as described in the following paragraphs. For

additional information about the files themselves, see Chapter 3, "The PASCAL Compiler".

1. Source File.

   The first prompt is for the name of the file that contains your source program:

```
Source filename [.pas]:
```

   The prompt reminds you that ".pas" is the default extension for the source filename. Unless the extension is something other than ".pas", you may omit it when you specify the filename.

   For now, enter "sort" followed by a carriage return (to indicate that the source file is 1:sort.pas).

2. Object File.

   The second prompt is for the name of the relocatable object file, which will be created during pass two:

```
Object filename [sort.obj]:
```

   The name in brackets is the name the compiler will give to the object file if you simply press the carriage return key at this point. The filename is taken from the source filename you gave in response to the first prompt; the ".obj" extension is the standard extension for object files.

   Respond to this prompt by presseing the carriage return key.

3. Source Listing File

   The third prompt is for the name of the source listing file, created during pass one:

```
Source listing [NUL.lst]:
```

   As before, the prompt shows the default. Because the source listing is not required for linking and executing the program, it defaults to a null file; that is, if you press the RETURN key, there will be no source listing file.

   However, if you enter any part of a file identifier (volume identifier or file name), the default extension is .lst, the default device is the current default drive, and the filename defaults to the name given to the source file.

For this session, assume that you want to send the source listing file to the terminal screen. Therefore enter "stdout" (for standard output) followed by a carriage return in response to the source listing prompt.

4. Object Listing File.

   The final prompt is for the object listing file, to be created during pass three:

```
Object listing [NUL.cod] -
```

   The null file is the default for the object listing, as it is for the source listing. IF you press the carriage return key, no intermediate files will be saved and you won't be able to run pass three. However, the same default naming rules apply here as elsewhere; if you enter any part of a file identifier, the default extension is ".cod", the default device is the current default drive, and the filename is the source filename.

   For now, enter "stdout" (for standard output) followed by a carriage return to request that the object listing be displayed on your terminal screen when you run pass three.

Compilation begins as soon as you have responded to all four prompts. The source listing is displayed on your screen, as requested. When pass one is complete, you should see the following message on your terminal screen.

```
Errors  Warns  In Pass One
     0      0
```

If the compiler had detected errors during compilation the value of Errors and Warns would be greater than zero.

The error and warning messages would appear in the source listing as it comes on your screen.

1. Errors are mistakes that prevent a program from running correctly.

2. Warnings indicate a variety of conditions, none of which will prevent the program from running, but which reflect poor programming practice or produce invalid results.

See Appendix H in the "M20 PASCAL Reference Manual" for a complete listing of messages and information about how to correct the errors in your program.

Pass one creates two intermediate files, "PASIBF.SYM" and "PASIBF.BIN". The compiler saves these two files on the default drive for use during pass two.

If there are errors, the two intermediate files are deleted and the remaining passes cannot be run. If pass one generates only warning messages, you can still run pass two and three, but you should go back and correct the source file at some point.

Pass Two

Remove the disk containing "pas1.cmd" from drive 0 and insert the disk containing the files "pas2.cmd" and "xtdata".

Start pass two by entering:

```
pas2
```

Respond 1 to the default drive question as you did for pass one. Pass two performs the following actions:

1. It reads the intermediate files "PASIBF.SYM" and "PASIBF.BIN" created in pass one.

2. It writes the object file.

3. It deletes the intermediate files created in pass one.

4. It writes two new intermediate files, "PASIBF.TMP" and "PASIBF.OID", for use in pass three. These files are written to the default drive.

When you are compiling your own programs, the last step described varies, depending on your response to the object listing prompt. If, as for this sample session, you plan to run pass three, pass two writes the two intermediate files. If in pass one you do not request an object listing, pass two writes and later deletes just one new intermediate file, "PASIBF.TMP".

When pass two is complete, a message like the following prints on your screen:

```
CODE AREA SIZE = #05EC  (1516)
CONS AREA SIZE = #00E6  ( 230)
DATA AREA SIZE = #0264  ( 612)
CASE TABLE AREA SIZE = #00C6  ( 198)

Pass Two    No Errors Detected.
```

The first four lines indicate the amount of space taken up by executable code (CODE), Constants (CONS), Variables (DATA), and case statements (CASE). The message concerning the number of errors refers to pass two only, not to the entire compilation.

Pass Three

Remove the disk containing "pas2.cmd" from drive 0 and insert the disk containing "pas3.cmd".

Start pass three by entering:

```
pas3
```

Respond 1 to the default drive prompt as you did in pass one and pass two. Pass three requires no other input. It reads "PASIBF.TMP" and "PASIBF.OID", the intermediate files created during pass two, and because of your earlier response to the object listing prompt, writes the object code listing to your screen.

When pass three is complete, the two intermediate files are deleted. If after requesting an object listing, you choose not to run pass three, you should delete these files yourself (to save space). The Table below is a summary of the files read and written by each of these three passes of the compiler during this sample session.

PASS	FILES READ	FILES WRITTEN	FILES DELETED
1	sort.pas PASKEY	stdout PASIBF.SYM PASIBF.BIN	
2	PASIBF.SYM PASIBF.BIN xtdata	sort.obj PASIBF.OID PASIBF.TMP	PASIBF.SYM PASIBF.BIN
3	PASIBF.OID PASIBF.TMP	stdout	PASIBF.OID PASIBF.TMP

See Chapter 3, "The PASCAL compiler" for details about compiler switches and other ways of responding to the compiler prompts.

LINKING YOUR M20 PASCAL PROGRAM

Now you are ready to link your program. Linking converts the relocatable object code into an executable program by assigning absolute addresses and setting up calls to the runtime library.

Remove the disk containing "pas3.cmd" from drive 0, and insert the disk containing "link.cmd", the link command file "paslnk", and the runtime library file "runtime.lib".

Start the linker by entering:

```
li command 0:paslnk input 1:sort.obj output 1:sort.cmd
```

The linker will create the executable program "sort.cmd" on drive one. For more detail see Chapter 4, "The LINK Command".

EXECUTING YOUR M20 PASCAL PROGRAM

When linking is complete, the operating system prompt returns. To run the sample program, just enter the first two characters of the file name (in this case "so") followed by a carriage return.

The command directs PCOS to load the executable file "sort.cmd", fix segment addresses to their absolute value (based on the address at which the file is loaded), and start execution. This concludes the sample session. Additional information on compiling and on linking is provided in Chapter 3, "The PASCAL Compiler" and Chapter 4, "The LINK Command" respectively.

SAMPLE SESSION LOG

The following page shows a log of the entire sample session. Your responses are shown in bold-face and a carriage return is indicated by **/CR/.**

```
0> pas1 /CR/
Ready diskette in default drive 0
or enter new default drive number: 1 /CR/
Source filename [.pas]: sort /CR/
Object filename [sort.obj]: /CR/
Source listing [NUL.lst]: stdout /CR/
Object listing [NUL.cod]: stdout /CR/

    [Source listing display]

    Errors  Warns  In Pass One
         0      0

1> pas2 /CR/
Ready diskette in default drive 0
or enter new default drive number: 1 /CR/

        CODE AREA SIZE = #05EC  (1516)
        CONS AREA SIZE = #00E6  ( 230)
        DATA AREA SIZE = #0264  ( 612)
        CASE TABLE AREA SIZE = #00C6  ( 198)

        Pass Two    No Errors Detected.

1> pas3 /CR/
Ready diskette in default drive 0
or enter new default drive number: 1 /CR/

     [Object listing display]

0> link command 0:paslnk input 1:sort.obj output 1:sort.cmd /CR/
0> so /CR/
```

3. THE PASCAL COMPILER

ABOUT THIS CHAPTER

In this chapter the three stages of the M20 PASCAL compiler are described in detail.

CONTENTS

STRING KEYWORDS

ENTRY

The ENTRY keyword may occur once. It provides a global symbol name which is to be made the entry point of the executable program. The entry point is determined as follows:

- If an ENTRY keyword is given, then the entry point specified is used, regardless of any definition within the input module itself.

- If no ENTRY keyword is given, then the entry point is set as defined in the input module.

The entry point of a PASCAL main program is located at the symbol "_begxqq".

MESSAGE

A MESSAGE keyword supplies the ASCII text (which must be one string) to go in the message record of the load file. There may be any number of MESSAGE keywords in one LINK command. The message record is the last record of the load file and does not form part of the executable program itself. It can be used for comments, remarks, date and time of operation, etc.

SIMPLE KEYWORDS

QUIET

The QUIET keyword causes output normally sent to the standard output to be suppressed, except for fatal error messages. If no QUIET keyword is given, the following information will be displayed:

- The LINK header line and version number.

- All error messages.

- A list of unresolved references.

VERBOSE

This keyword causes extra information to be sent to standard output. The command line being executed is displayed, entry to each new module is noted, and a warning is issued each time the possibility of an error is encountered.

STATISTICS

The STATISTICS keyword, if specified, causes the program to output statistics on how much of LINK's memory was used.

OPTIMIZE

Specifying the OPTIMIZE keyword in the command line causes the output file to be optimized by not including uninitialized memory at the beginning or the end of the program text section of the output load file. This produces a smaller load file and saves time in loading the program into memory.

BLOCK KEYWORD

BLOCK

The parameters of a BLOCK keyword are names of program sections that are to be loaded in one contiguous region of memory (i.e. a block). The BLOCK keyword may occur any number of times on a LINK command line. The program sections can also be specified by patterns with the use of the following Wild Card characters;

- An asterisk (*) which matches any string.
- A question mark (?) which matches any single character.
- [ab...] which matches any single character in the square brackets.
- [a-b] which matches any single character in the interval a-b

A pattern stands for all the section names which match that pattern, and which have not been used previously in the current or any other block. The sections are taken in the same order that they occur in the input object modules.

If a section does not fit in the first block that it matches, a warning message is issued by LINK, and the section is left as a candidate for other blocks that it also matches. Any sections which remain unplaced are reported via a warning message and ignored thereafter.

In the absence of a BLOCK keyword, "BLOCK *" is assumed by default as the last keyword on the command line. This means that LINK will attempt to place all sections in one block the size of which is defined in the command line (see BLOCKSIZE). If a program does not fit in one block then two or more BLOCK keywords need to be specified for a successful LINK operation.

Section names in a PASCAL program are created by the PASCAL compiler. A

program with program name "progname" will contain:

- a (program) code section with section name "progname_p"

and, as the case may be, any combination of the following:

- a CASE statement table section with section name "progname_c"
- a data section with section name "progname_d"
- a constant definition section with section name "progname_k"
- a stack section with section name "progname_s

A necessary condition for PASCAL programs to execute is that all stack sections must be placed in the same block. You can ensure this by specifying "BLOCK *_s" in a LINK command line where two or more BLOCK keywords are used.

KEYWORD ORDER

The order in which keywords appear has no gross effect on the outcome of the operation. The effects of ordering are due to the fact that files are opened and flags are set when their respective keywords are encountered. For example, keywords which appear before the MAP or the VERBOSE keyword do not get echoed into the MAP file, or on standard output. The relative order of the BLOCKSIZE and BLOCKTYPE keywords is important because their parameters are used as default values for subsequently defined blocks.

ERRORS

If any fatal error occurs during the parsing of keywords or the execution of the locate operation, the program is stopped immediately with an error message on standard output and, if it was specified, the map file.

ALTERNATE STACK/HEAP SEGMENTS

Compiled code output by the M20 PASCAL compiler assumes that the stack and heap occupy a contiguous block of memory in a single segment. In addition, this stack/heap block must be statically allocated so that the loader can resolve relocatable references to it at load time.

The runtime library contains a 4k byte declaration for the stack/heap block. Alternative declarations for the block are provided in separate object files included in the PASCAL package on the same diskette which

contains the runtime library. The files are the following:

FILE NAME	STACK SIZE (in bytes)
hpstk2k.obj	2k
hpstk8k.obj	8k
hpstk16k.obj	16k
hpstk32k.obj	32k
hpstk64k.obj	64k

By specifying any one of these files as an input module to LINK, along with the rest of the object files, the library declaration will be ignored. The size of the resulting load file is not affected by these alternative declarations. the memory requirements for the program, however, will vary accordingly. Using the 2k byte stack/heap will reduce the memory required by a program but may limit its dynamic capabilities. The larger stack/heap blocks, on the other hand, may give a program more dynamic capability but will increase its memory requirements.

PLOADING PASCAL PROGRAMS

PASCAL programs output by LINK may be PLOADed but may be executed only once after PLOADing. This is because some data is initialized when the program is loaded and then changed during execution. It is not possible to execute a PLOADed program a second time because the initialized data would not have the correct initial value.

The possibility to PLOAD PASCAL programs is still useful, however, because it leaves both disk drives available to the program on an M20 with two disk drives. It is even more useful on an M20 with a single disk drive. Attempting to execute a PLOADed PASCAL program a second time will return an error.

Example

The following LINK command will create an executable file "myprog.cmd" from the object file created in the example shown in chapter 3, "myprog.cmd" The command will also create a map file "myprog.map". It is assumed that the program uses the M20 PASCAL library ("interfs.lib"), the PASCAL graphics library ("interfg.lib") and consequently the graphics package library "graph.lib".

```
li map 1:myprog.map,input 1:myprog.obj,library 0:runtime.lib
0:interfs.lib 0:interfg.lib 0:graph.lib,entry _begxqq,block *_s
block *,output 1:echo.cmd /CR/
```

The same result can also be obtained by specifying the command file shown below in the following command:

```
li command 1:comlist /CR/
```

On the following page is a listing of the file "comlist"

```
!          Command file for LINKing   myprog.cmd          !

                        MAP 1:myprog.map

! Create a map file "myprog.map" on  the  disk  inserted  in !
! drive 1. Note that as this is the  first  keyword  in  the !
! file all that follows will appear in the map file.         !

                       INPUT 1:myprog.obj

! If more than one  file  need  to  be  specified  these can !
! follow even on successive lines as long as  there  are  no !
! intervening keywords.                                      !

                     LIBRARY 0:runtime.lib
                             0:interfs.lib
                             0:interfg.lib
                             0:graph.lib

! LINK will pick from these four library files  the  modules !
! that are referenced by the input program.                  !

                        ENTRY _begxqq

! The symbol _begxqq specifies the entry point of  a  PASCAL !
! program.                                                   !

                          BLOCK *_s

! This keyword will group all stack sections  in  one  block !

                           BLOCK *

! This keyword will group  all  the  remaining  sections  in !
! another block.                                             !

                       OUTPUT 1:myprog.cmd

!          Only one output file can be specified             !
```

Fig. 4-2 Listing of the File "comlist"

5. LIBRARIES

ABOUT THIS CHAPTER

This chapter describes the use of libraries and the MLIB command for creating library files.

CONTENTS

INTRODUCTION

It is common programming practice to use a library of subroutines to be made available to a series of programs. Mathematical programs, for instance might use a library of subroutines for calculating trigonometric functions, and text oriented programs might use a library of string comparison functions.

PASCAL procedures and functions can be compiled as separate routines. These routines can then be used by a separate program where they are declared as external procedures and/or functions as the case may be. There are two ways in which external functions and/or procedures can be LINKed to the main program;

1. you can specify the external functions and/or procedures as input object files along with the main program, or

2. group the external functions and/or procedures in a library file and specify the library file using the LIBRARY keyword in a LINK operation.

When LINK discovers an external variable which is not present in any input file, then, if the LIBRARY keyword was specified, it will search through the list of library file(s) (specified after the LIBRARY keyword) for a "global" definition. Once the subroutine name is found, the module containing the subroutine is incorporated into the output load file. Only the modules referenced by input files are included in the output load file along with the rest of the input modules. A library module "Y" referenced by another library module "X" in the same library file will only be included if "X" is located before "Y" in the library.

When LINKing a program using either library files, or more than one input file it is important to ensure that the program entry point is well defined (see the ENTRY keyword in chapter 4 "The LINK Command").

Library files can be created using the MLIB command described below.

MLIB

Creates a library file of object modules from a group of object files.

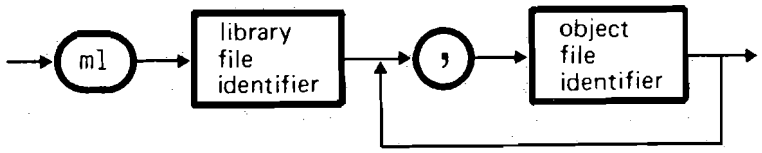


Fig. 5-1 The MLIB Command

Where

SYNTAX ELEMENT	MEANING
library file identifier	The name of the file that is to contain all the object modules in the specified object files. This must be complete with any necessary volume identifier and/or file password. The file will be created if it does not exist or, if it already exists, it will be overwritten with the new output. A library file is usually assigned the extension ".lib".
object file identifier	The name of an object file complete with any necessary volume identifier and/or file password. You can use the two PCOS wild card characters (*) and (?) to specify more than one file; an asterisk (*) matches any string and a question mark (?) matches any one single character.

Characteristics

During execution the MLIB command needs to create a temporary work file on the disk inserted in the last selected disk drive. This means that MLIB will not execute if called from a write protected diskette. Rather than remove write protection from the diskette it is recommended to PLOAD the MLIB command or to copy the file "mlib.cmd" from your copy protected diskette onto the disk where you want to create your library files, or some other disk.

Example

IF you enter	THEN
ml 1:pascal.lib,1:prog1.obj, 1:prog2.obj /CR/	the file "pascal.lib" is created on the diskette inserted in drive 1. This file will contain all the object modules contained in the object files "prog1.obj" and "prog2.obj" both of which are resident on the same diskette inserted in drive 1.

M20 LIBRARIES

The M20 PASCAL package includes three libraries, the M20 PASCAL library, the PASCAL graphics library and the runtime library. These are contained in the three files "interfs.lib", "interfg.lib" and "runtime.lib" respectively.

The M20 PASCAL library is a library of functions, and one procedure which interface the M20 System Calls, while the PASCAL graghics library contains procedures and functions which interface the M20 Graphics Package. The Graphics Package itself is a library of graphics routines called "graph.lib", these routines however cannot be accessed directly by a PASCAL program.

The runtime library contains the modules which implement real number arithmetic. This library must be specified with all PASCAL programs in the LINK phase.

The functions (and one procedure) of the M20 PASCAL library are introduced in chapter 6 and all the descriptions are in chapter 7. As for the Graphics library functions and procedures these are introduced in chapter 8 and detailed in chapter 9.

USING THE M20 PASCAL LIBRARY

To use any of the functions (or one procedure) in this library, you have to declare the function (or procedure) as EXTERNAL. Subsequently, when LINKing you must specify the library file "interfs.lib" using the LIBRARY keyword. You must also ensure that the entry point of the program is well specified (see the ENTRY keyword in chapter 4 "The LINK Command").

USING GRAPHICS

The functions and procedures of the PASCAL graphics library must be declared as EXTERNALs before they can be used in a PASCAL program (as with the M20 PASCAL library). In the LINK stage however both the PASCAL graphics library "interfg.lib" and the Graphics Package library "graph.lib" must be specified with the LIBRARY keyword. Here again you have to make sure that the entry point of the program is well specified (see the ENTRY keyword in chapter 4 "The LINK Command").

6. INTRODUCTION TO THE M20 PASCAL LIBRARY

ABOUT THIS CHAPTER

This chapter is a general description of the functions and one procedure of the M20 PASCAL library. The library is divided in functional groups and the characteristics of each group are discussed.

CONTENTS

INTRODUCTION

The next two chapters describe the PASCAL functions and one PASCAL procedure which interface some of the M20 System Calls. System Calls are PCOS procedures used to interface with I/O or to manage memory.

When any one of the PASCAL functions (or procedure) in the M20 PASCAL library is called, the specified parameters (if there are any) are passed as register assignments and the system call it interfaces is executed. For this reason all the parameter values are such that they can be expressed in a maximum of eight or sixteen bits. In some cases, where strings or large data structures are to be passed to the call, these have to be stored in memory first and the memory addresses of these structures are passed as parameters instead.

FUNCTION RESULTS

All the functions in the PASCAL library will return the value zero if and only if there are no errors. If there is an error the function will return the decimal error code.

A convenient way to use these functions in a PASCAL program is in an assign statement, for example:

```
err := functionidentifier (parameter1,parameter2,...,parameterN)
```

You can then check the variable "err" for any errors, and, if "err" is non-zero, you can pass it as a parameter to the "error" procedure ("error" is the one and only procedure in the M20 PASCAL library).

FUNCTIONAL GROUPS

The M20 PASCAL library can be divided into functional groups as follows:

- Bytestream I/O functions
- Block transfer functions
- Storage Allocation functions
- Time and Date functions
- IEEE 488 functions

- Error procedure

- Miscellaneous functions (and one procedure)

In this chapter we shall discuss these functional groups. In the next chapter the functions are detailed in alphabetical order.

BYTESTREAM I/O FUNCTIONS

Bytestream functions are used for:

- Transferring bytes of data to or from an I/O device

- Sending control information to a device or to a device driver

- Receiving status information from a device

The following is a list of bytestream I/O funtions used to interface with the disk, the printer, the RS-232 communications port, and the console (keyboard and video).

closedevice	peof
closefile	pseek
directory	putbyte
getbyte	readbytes
getlen	readline
getposition	remove
getstatus	rename
lookbyte	resetbyte
opendevice	setcontrol
openfile	writebytes

DID (Device IDentifier) Numbers

In most of these functions the file or device is identified by a DID (Device IDentifier) number. The operating system maintains a table associating DIDs with file and device pointers. Opening a disk file or a device creates a stream data sructure and places a pointer to it in the device pointer table. Closing the disk file (or device) sets this pointer to nil and releases any associated table space.

Some devices or files are always open. For example, the keyboard and the screen (the default window) are always open. They can, however, be closed and re-opened by using the PCOS Device Rerouting feature.

A table of DID assignments is included in the Appendix.

BLOCK TRANSFER FUNCTIONS

The block transfer functions allow the programmer to set memory to a fixed value, or to transfer data from one segment to another, or to clear memory.

The block transfer functions are the following:

bclear	bset
bmove	bwset

STORAGE ALLOCATION FUNCTIONS

It is possible for a user program to call allocate or release heap space.

Functions which open or close a disk file use the system calls interfaced by these functions internally to allocate or release heap space.

The following are the Storage Allocation functions:

maxsize	newsameseg
newabsanyseg	pdispose
newabsolute	pnew
newlargestblock	stickynew

TIME AND DATE FUNCTIONS

The M20 system has a real-time clock which maintains both date and time. This clock must be reset each time the system is powered up.

Time or date setting are done by passing the address of an ASCII string to the operating system. Likewise, the time or the date can be read by transfering an ASCII string from the operating system.

These operations are performed by the following functions:

getdate	setdate
gettime	settime

IEEE-488 FUNCTIONS

Using the functions in this group you can perform the following operations on an IEEE-488 bus:

- Control the IFC (interface clear) and REN (remote enable) lines;
- Receive a service request from another device on the bus, identifying the requesting device through serial polling, and processing the service request;
- Write control bytes (e.g.: "Device Clear", "Device Trigger", etc.) to other devices;
- Address, write data to, and read data from, other devices; and
- Allow the devices within an IEEE-488 network to transfer data on the bus (i.e.: assign "Talker" status to one device, and "Listener" status to one or more devices).

The IEEE-488 functions are the following:

ilninput	iset
ipoll	isrq0
iprint	isrq1
iread	iwrite
ireset	

The errors which can be returned by these functions are particular to these functions only. These errors are explained in the table below.

ERROR CODE	MEANING
3	Invalid termination of input bytestream. The two valid cases are: 1. the number of data bytes received equals the value specified. The last data byte is accompanied by the END condition (EOI true, ATN false). 2. the number of data bytes received equals the value specified. The last data byte is followed by a CR, LF pair with the END condition accompanying the LF.

9	Talker or Listener address greater than (hex) 1F.
10	IEEE board not present.
11	15 second polling loop (for "byte in", "byte out", or "input buffer empty" condition) timed out; handshake could not be completed within 15 seconds.

For further details on the IEEE-488 interface see the "M20 I/O with External Peripherals User Guide", however note that this manual describes the BASIC interface to the system calls which are interfaced by the IEEE-488 functions in the M20 PASCAL library.

ERROR PROCEDURE

The "error" procedure is the only procedure in the M20 PASCAL library. This procedure will display on the screen the error message associated with a specified error code.

MISCELLANEOUS FUNCTIONS

The following miscellaneous functions complete the list of contents of the M20 PASCAL library:

bootsys
checkvol
crlf
dhexbyte
dhexlong
dhexword
diskfree
dlong
dnumw
dstring
getvol
parsename
sdevtab
search
setsysseg
setvol
stringlen

7. THE M20 PASCAL LIBRARY

ABOUT THIS CHAPTER

This chapter details all the functions and one procedure of the M20 PASCAL library. The descriptions follow each other in [illegible]ical order.

CONTENTS

bclear

Sets a specified block of memory to zero.

Function Declaration

```
FUNCTION bclear ( start  : ADSMEM ;
                  length : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function sets a block of bytes, of a specified size, and starting at a specified address, to zero. The first input variable "start" is a segmented pointer to the first byte of memory to be set,while the second input variable, "length" is the number of bytes to be set to zero.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

bmove

Moves a block of bytes from one location to another.

Function Declaration

```
FUNCTION bmove ( start       : ADSMEM ;
                 destination : ADSMEM ;
                 length      : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function moves a block of bytes, of specified length, and starting at a specified address, to a specified destination block. The input variable "start" is a segmented pointer to the first byte of memory to be moved, while the input variable "length" is the number of bytes to be moved. The input variable "destination" is a segmented pointer to the first byte of the destination memory block.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Reboots (initializes) the system.

Function Declaration

```
FUNCTION bootsys : INTEGER ;
```

Characteristics

This function is used to reboot the system exactly as does pressing the blue shift plus reset keys. In other words, the system reboots, but bypasses the diagnostic checks.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Sets a block of bytes to a specified value.

Function Declaration

```
FUNCTION bset ( value   : BYTE    ;
                start   : ADSMEM  ;
                bytelen : WORD    ) : INTEGER ; EXTERN ;
```

Characteristics

This function sets each byte of a block of memory to the value of the input parameter "value". The input variable "start" is a segmented pointer to the first byte of memory to be set, while "bytelen" is the number of bytes to be set.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Sets a block of words to a specified value.

Function Declaration

```
FUNCTION bwset ( wvalue  : WORD   ;
                 start   : ADSMEM ;
                 wordlen : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function sets each word of a block of memory to the value of the input parameter "wvalue". The input variable "start" is a segmented pointer to the first word of memory to be set, while "wordlen" is the number of words to be set.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

checkvol

Forces a check of disk volumes

Function Declaration

```
FUNCTION checkvol : INTEGER ;
```

Characteristics

There are no input parameters for this call. All volumes are forced to read their verification codes on their access.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

closedevice

Closes the specified device.

Function Declaration

```
FUNCTION closedevice ( did : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function disables the hardware interrupts, closes the specified device and then releases both buffer and table space. The input "DID" identifies the device.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

19, 25, 26	Com (RS-232-C), Com1, Com2

closefile

Closes the specified disk file.

Function Declaration

```
FUNCTION closefile ( did : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function closes the specified file and then releases both buffer and table space. The input "DID" identifies the file.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
20 - 24	disk files (PCOS)

Does a CR and a LF.

Function Declaration

```
FUNCTION crlf : INTEGER ;
```

Characteristics

This function will do a carriage return and a line feed. There are no parameters.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

dhexbyte

Displays a byte in Hex.

Function Declaration

```
FUNCTION dhexbyte ( byteval : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

The byte supplied in the input parameter "byteval" is displayed as two hex digits.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Displays a long word in hexadecimal.

Function Declaration

```
FUNCTION dhexlong ( longval :INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

The long word supplied in the input parameter "longval" is displayed as eight hex digits.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

dhexword

Displays a word in hex.

Function Declaration

```
FUNCTION dhexword ( wordval : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

The word supplied to the input parameter "wordval" is displayed as four hex digits.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

directory

Displays a list of files from a specified disk.

Function Declaration

```
FUNCTION directory ( fileadr : ADSMEM ;
                     filelen : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function is used only for files. It lists the contents of the directory of the specified volume, on the current window of the M20 screen. The input variable "filelen" is the number of bytes in the file identifier. The input "fileadr" is the address of the file identifier. The file identifier may contain a volume identifier and/or wild card characters ("*" and "?").

The display lists the names of the specified files on the specified (or default) volume in compact form.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

diskfree

Returns the number of free sectors on the disk.

Function Declaration

```
FUNCTION diskfree (    volnum : WORD     ;
                   VAR secnum : INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns into "secnum" the number of sectors that are available for use on the specified disk. The input parameter "volnum" is the number of the disk drive where the volume to be checked is inserted. (use -1 for the current volume).

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Displays a number as an unsigned decimal integer.

Function Declaration

```
FUNCTION dlong ( longval :INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

The hexadecimal number supplied in the input parameter "longval" is displayed as an unsigned decimal integer, left-justified with leading zeroes omitted.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

dnumw

Displays a number as an unsigned decimal integer.

Function Declaration

```
FUNCTION dnumw ( wordval : INTEGER ;
                 width   : WORD    )  : INTEGER ; EXTERN ;
```

Characteristics

The hexadecimal number in the input parameter "wordval" is displayed as an unsigned decimal integer. The input parameter "width" specifies the field width for display.

The display is right-justified in the field, with leading zeroes changed to spaces.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Displays a string message.

Function Declaration

```
FUNCTION dstring ( stringadr : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function displays a string message stored in the address pointed to by the input parameter "stringadr". The string must be terminated with a null (0) byte.

The message may include any number of carriage returns, but note that a linefeed will be automatically displayed after each carriage return in the string.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

error

Displays a standard error message.

Procedure Declaration

```
PROCEDURE error ( parnum    : BYTE  ;
                  errorcode : BYTE  ) ; EXTERN ;
```

Characteristics

This procedure is only called if there are errors. The procedure displays the message "Error nn" in parameter "xx" where "nn" is one of the standard error codes (in decimal) specified by the hexadecimal value of the input parameter "errorcode", and xx is the parameter number specified in the input parameter "parnum". If "parnum" is zero then only the message 'Error nn' is displayed.

Note:

If the EPRINT command is resident, then an error message will also be displayed.

getbyte

Returns the first byte from a designated device, removing the byte from the device buffer.

Function Declaration

```
FUNCTION getbyte (    did     : WORD ;
                  VAR retbyte : BYTE ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns the first byte in the input buffer (from file or designated device) and places that byte in the variable "retbyte". The DID is an integer which identifies the source of the input. Valid DID numbers are listed below.

In the case where the DID is either 17 or 19, if the input device buffer is empty, the system will wait until a byte is input and available in the buffer before returning to the caller with the byte in the variable "retbyte".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
17	console
20 - 24	disk files (PCOS)
19,25,26	Com (RS-232-C), Com1, Com2

getdate

Returns the system date at a specified address.

Function Declaration

```
FUNCTION getdate ( dataadr : ADSMEM ;
                   length  : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This call returns the ASCII string giving the system date at a specified data address. The input variable the "dataadr" is the address where the ASCII string will be stored, while "length" is the maximum length of the string, which is stored in the BASIC data area.

The format of the returned date is:

```
dd:mm:yyyy
```

where "dd" is the day, "mm" is the month, and "yyyy" is the year.

There will be leading zeroes to make each field two characters in length and the character separating the various fields for the date will be that used in the last call to function setdate. The system initializes the separator character to ":".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

getlen

Returns either the length of a file or the number of bytes in the input buffer.

Function Declaration

```
FUNCTION getlen (    did    : WORD     ;
                  VAR length : INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

DEVICES This function returns into the variable "length" either the number of bytes bytes currently in the input buffer (for a device) or the file length (in bytes). The input DID identifies the device or file.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
20 - 24	disk files (PCOS)
19,25,26	Com, Com1, Com2

getposition

Gets the position of the next byte to be read or written.

Function Declaration

```
FUNCTION getposition (    did       : WORD     ;
                      VAR fposition : INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

This call returns into the variable "fposition" the position, in bytes, of the next byte to be read or written. The input "DID" identifies the file. A list of valid DID numbers is given below.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
20 - 24	disk files (PCOS)

getstatus

Reads a single word from the Device Parameters Table.

Function Declaration

```
FUNCTION getstatus (     did     : WORD ;
                         wordnum : WORD ;
                     VAR parword : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function allows a single word to be read from the Device Parameter Table (see appendix E). The input "wordnum" is the word number to be read. The word read from the Device Parameter Table is returned in the variable "parword".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

19, 25, 26	Com(RS-232-C), Com1, Com2

gettime

Returns the system time at a specified address.

Function Declaration

```
FUNCTION gettime ( dataadr : ADSMEM ;
                   length  : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns the ASCII string giving the system time at a specified data address. The input variable "dataadr" is the address where the ASCII string will be stored, while "length" is the maximum length of the string, which is stored in the BASIC data area.

The format of the time returned is:

```
hh:mm:ss
```

where "hh" is the hour (in 24-hour time), "mm" is minutes, and "ss" is seconds.

There will be leading zeroes to make each field 2 characters in length, and the character separating the various fields for the time will be that used in the last call to function settime. The system initializes the separator character to ":".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

getvol

Returns the current default volume number.

Function Declaration

```
FUNCTION getvol (    vbuffer : ADSMEM ;
                     bufsize : WORD   ;
                 VAR vsize   : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns the current default volume identifier in the buffer pointed to by the input variable "vbuffer". The size of this buffer is specified in the variable "bufsize". The actual size of the volume identifier string is returned in "vsize".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

ilninput

Places bytes received into a buffer as a single line of data.

Function Declaration

```
FUNCTION ilninput (     buffer          : ADSMEM ;
                        buflen          : WORD   ;
                        talkeradr       : WORD   ;
                        listenadr       : WORD   ;
                    VAR bytes_not_read  : WORD   ) : INTEGER ;
                                                     EXTERN  ;
```

Characteristics

This function will transfer bytes received sequentially from a driver specified by "talkeradr" to a buffer specified by the pointer "buffer", and, if a valid device address is specified in "listenadr", to a listener.

The input "buflen" is the length of the buffer. The difference between the buffer length specified and the number of bytes read will be returned in the variable "bytes_not_read".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

ipoll

Polls a specified device on an instrument bus.

Function Declaration

```
FUNCTION ipoll (    talkadr    : WORD    ;
                 VAR statusptr : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function polls the device specified, within a serial service request poll. The input variable "talkadr" identifies the device.

The function tests the device address, reads the device status byte, and saves it in an address pointed to by the return variable "statusptr".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

iprint

Transfers data from a buffer to a listener address.

Function Declaration

```
FUNCTION iprint ( buffer    : ADSMEM  ;
                  listenadr : WORD    ;
                  buflen    : WORD    ;
                  delimiter : WORD    )  : INTEGER ; EXTERN ;
```

Characteristics

This function transfers data sequentially from a buffer, specified by the memory address in the input variable "buffer", and by its length in input variable "buflen", to a listener address specified by the input variable "listenadr".

The input parameter "delimiter" determines the data-stream delimiter. Zero specifies "END" as the data-stream delimiter, and one specifies "CR, END" as the data-stream delimiter sequence.

If there are any output bytes for transfer, they will be written to bus, with ATN false.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

Outputs commands (optional) and reads data bytes (optional).

Function Declaration

```
FUNCTION iread ( comlist : ADSMEM  ;
                 comlen  : WORD    ;
                 buffer  : ADSMEM  ;
                 buflen  : WORD    )  : INTEGER ; EXTERN ;
```

Characteristics

If there is a command list, this function will assert ATN and output commands. It then reads a specified number of bytes, and places them sequentially in a buffer.

The input parameter "comlist" points to the address of the command list. This list, if present, is stored as a sequence of bytes, 2 to the word.

The input "comlen" is the command list length in the 15 low order bits. The high order bit is always zero (0). If there is no command list then "comlen" is specified as zero.

The input "buffer" points to the buffer which will receive the data bytes, while input "buflen" is the number of bytes to be read.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

ireset

Causes the remote enable (REN) message to be sent false.

Function Declaration

```
FUNCTION ireset : INTEGER ;
```

Characteristics

This function causes the remote enable (REN) message to be sent false.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

iset

Causes a remote enable (REN) or an interface clear (IFC) to be sent.

Function Declaration

```
FUNCTION iset ( operand : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function causes the remote enable (REN) message or the interface clear (IFC) pulse to be transmitted, depending upon the value of the input variable "operand".

If "operand" is zero, then the REN message is sent true; if one is specified, then the IFC pulse is sent.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

isrq0

Disables the service request (SRQ) interrupt.

Function Declaration

```
FUNCTION isrq0 : INTEGER ;
```

Characteristics

When using the IEEE-488 package this call disables the service request (SRQ) interrupt. (For more information see function isrq1.)

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

Enables the service request (SRQ) interrupt.

Function Declaration

```
FUNCTION isrq1 : INTEGER ;
```

Characteristics

This function will set the global flag "srq_488" (byte) to one when an SRQ interrupt occurs. (This flag is stored in the mailbox area, see appendix G)

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

iwrite

Outputs commands (optional) and writes data bytes (optional).

Function Declaration

```
FUNCTION iwrite ( comlist : ADSMEM  ;
                  comlen  : WORD    ;
                  numadr  : ADSMEM  ;
                  numlen  : WORD    )  : INTEGER ; EXTERN ;
```

Characteristics

If there is a command list, this function will assert ATN and output commands. If there are any data bytes to be output, writes them to bus with ATN false.

The input parameter "comlist" points to the address of the command list. This list, if present, is stored as a sequence of bytes, two to the word. The input "comlen" is the command list length in the 15 low-order bits of the word; the high-order bit is set to one to specify "END" as the data-stream delimiter, and zero to specify "CR, END" as the data-stream delimiter sequence. If there is no command list then "comlen" is specified as zero.

The input "numadr" points to the address of the list of numeric values. It, too, is stored as a sequence of bytes, two to a word. The input "numlen" is the length of the list of numeric values. If there are no data bytes to write then "numlen" is specified as zero.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code. The error codes which can be returned by this function (or other IEEE-488 functions) are explained in chapter 6 in the section on IEEE-488 functions.

lookbyte

Returns the next byte from the designated device buffer without removing the byte from the buffer.

Function Declaration

```
FUNCTION lookbyte (    did       : WORD   ;
                   VAR retbyte   : BYTE   ;
                   VAR bufstatus : BYTE   ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns the first byte of a device input buffer (undefined if none) in the variable "retbyte", without removing it from the buffer. The DID is an integer, identifying the device. Valid DIDs are listed below. Also returned is the status of the device buffer in the variable "bufstatus", this will be FF if the buffer is not empty or 00 otherwise.

Note:

Ring buffers are maintained for the interrupt driven input devices. Characters are placed into the buffers immediately as they are received and are available to programs via the two functions "lookbyte" and "getbyte".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

17	console
19,25,26	Com (RS-232-C), Com1, Com2

maxsize

Returns maximum free heap size

Function Declaration

```
FUNCTION maxsize ( VAR size : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns in the variable "size" the largest free heap block in the current segment. Size is returned in bytes. By default the current segment is segment 2 unless the program has executed a "newabsanyseg" function, in which case the segment number is that specified in the most recent "newabsanyseg".

A simple way to change the current segment number for a program is to do a "newabsanyseg" function with the input variable "size" specified as zero.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Allocates a block at a specified address.

Function Declaration

```
FUNCTION newabsanyseg (VAR tableadr : ADSMEM ;
                           size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function is similar to function "pnew" except that the block allocated will be at a specified address. The input address is specified in the variable "tableadr". The input parameter "size" is the number of bytes requested (this must be an even number).

After execution "tableadr" will contain the address of the actual block allocated. If the requested value is too close to the end of a previous block, the actual value may be two bytes lower than the requested value, but will still include the requested length. If the space requested is not available, a nil-pointer (hex FFFFFFFF) will be returned in "tableadr", but no error code will be returned by the function.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

newabsolute

Allocates a block at a specified address.

Function Declaration

```
FUNCTION newabsolute (VAR tableadr : ADSMEM ;
                          size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function is similar to function "newsameseg" except that the block allocated will be at a specified address. The input address is specified in the variable "tableadr". The input parameter "size" is the number of bytes requested (this must be an even number).

After execution "tableadr" will contain the address of the actual block allocated. If the requested value is too close to the end of a previous block, the actual value may be two bytes lower than the requested value, but will still include the requested length. If the space requested is not available, a nil-pointer (hex FFFFFFFF) will be returned in "tableadr", but no error code will be returned by the function.

This function allocates blocks in the current segment. By default this is segment 2 unless the program has executed a "newabsanyseg" function, in which case the segment number is that specified in the most recent "newabsanyseg".

A simple way to change the current segment number for a program is to do a "newabsanyseg" function with the input variable "size" specified as zero.

Note that this function is a subset of function "newabsanyseg". It has been maintained for compatibility with preceding releases.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

newlargestblock

Allocates the largest block of bytes from heap.

Function Declaration

```
FUNCTION newlargestblock (VAR tableadr : ADSMEM ;
                          VAR size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function allocates a the largest free block in memory, returning in "tableadr" a pointer to the location of the first byte of the block and the length of that block in the variable "size".

If the block cannot be allocated, "tableadr"will contain a nil (hex FFFFFFFF) but no error code will be returned by the function.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

newsameseg

Allocates a block of bytes from heap in the current segment.

Function Declaration

```
FUNCTION newsameseg (VAR tableadr : ADSMEM ;
                         size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function allocates blocks in the current segment. By default this is segment 2 unless the program has executed a "newabsanyseg" function, in which case the segment number is that specified in the most recent "newabsanyseg".

A simple way to change the current segment number for a program is to do a "newabsanyseg" function with the input variable "size" specified as zero.

The input variable "size" is the size of the number of bytes to be allocated, and the returned variable "tableadr" is a pointer to the first byte of the allocated block.

Note that this function is a subset of function "pnew". It has been maintained for compatibility with preceding releases.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Opens a specified device.

Function Declaration

```
FUNCTION openfile ( did : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function opens the device specified by input variable "did".

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

19,25,26	Com (RS-232-C), Com1, Com2

openfile

Opens a specified file for read, write, etc.

Function Declaration

```
FUNCTION openfile ( did       : WORD   ;
                    mode      : WORD   ;
                    extentlen : WORD   ;
                    filelen   : WORD   ;
                    buffer    : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function will open the designated file in the mode specified by the input parameter "mode" as shown below:

0: Read, always from current position.

1: Write, always placing a new end of file.

2: Read/Write, allocating sectors beyond old EOF.

3: Append, seeks to end upon open, and then writes.

A file that does not exist cannot be opened in the read mode. A non-existent file, if opened by write or read/write, will be created. If it does exist, write mode will write over the old file.

If an existing file has been opened in the read/write mode, the user can then position the file pointer to its end to extend it using the function "pseek". However, Append mode does this automatically and then operates the same as the write mode.

The input "extentlen" designates the number of sectors to be allocated if the file is to be created. The request should always be one sector larger then the data requirements. If zero is specified, the number of sectors will be the default value (8). The input DID number identifies the file (see list below).

The input "filelen" is the number of characters in the file identifier (complete with an optional volume identifier and/or password).

The input "buffer" points to the file identifier string.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
20 - 24	disk files (PCOS)

parsename

Parses a file or a volume name.

Function Declaration

```
FUNCTION parsename (     stradr  : ADSMEM ;
                         strlen  : WORD   ;
                         nameptr : ADSMEM ;
                    VAR  vol     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function takes a file identifier of the form

```
" volname '/' volpswd ':' filename '/' filepswd "
```

and parses it into its various components. A drive unit is acceptable as volname .

Each component is placed into the appropriate compartment of the names record as follows:

```
volname  : 14 byte array
volpswd  : 14 byte array
filename : 14 byte array
filepswd : 14 byte array
```

The input parameter "stradr" is an address which points to the file identifier string. The length of the file identifier string is specified in "strlen" (this also includes the volume identifier if specified).

The input parameter "nameptr" is an address which points to the names record.

The returned parameter "vol" will contain the drive number where the disk which contains the file or volume name is inserted.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Releases heap space.

Function Declaration

```
FUNCTION pdispose ( VAR tableadr : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function releases memory space. The start address from where heap space is to be released is specified in the variable "tableadr". After execution "tableadr" will be a nil pointer.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

peof

Checks if an input character is available from a device.

Function Declaration

```
FUNCTION peof (    did       : WORD    ;
                VAR retstatus : BOOLEAN ) : INTEGER ; EXTERN ;
```

Characteristics

The function "peof" (end of file) will return a "false" value in "retstatus" if input character is available from the selected device. "retstatus" will be set to "true" in each of the following cases:

1. The selected file is not open.
2. The file is open for output only.
3. The console has been selected but no key has been struck.
4. The end of the disk file has been reached.

The input "DID" identifies the device; valid DIDs are listed below.

RS-232-C

For use with the RS-232-C, this function returns a "false" value in "retstatus" if the input buffer is not empty. "retstatus" will be "true" if the buffer is empty.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
17	console
19,25,26	Com (RS-232-C), Com1, Com2

pnew

Allocates a block of bytes from heap.

Function Declaration

```
FUNCTION pnew (VAR tableadr : ADSMEM ;
                   size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function allocates a block of bytes from the heap, returning in "tableadr" a pointer to the location of the first byte of the block.

The input variable "size" is the number of bytes to be allocated.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

pseek

Positions a file pointer as specified.

Function Declaration

```
FUNCTION pseek ( did       : WORD     ;
                 fposition : INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

This function will position the file pointer for the specified stream (opened file) to the position specified. The input "DID" identifies the device. The input "fposition" is a 32-bit pointer. Position zero is the first byte.

Seeking past the EOF while the file is open for read/write will automatically allocate new sectors.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
20 - 24	disk files (PCOS)

putbyte

Transmits a byte to a specified device.

Function Declaration

```
FUNCTION putbyte ( did    : WORD ;
                   inbyte : BYTE ) : INTEGER ; EXTERN ;
```

Characteristics

This function transmits the byte supplied in "inbyte" to the device or file specified by the DID. Valid DIDs are identified below. For files, no information is returned about the validity or EOF state of the DID.

If the device is the RS-232-C port, and the port is not ready to send, the driver will wait for a timeout period and then the function will return the appropriate error code if nothing is sent.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
17	console
18	printer
20 - 24	disk files (PCOS)
19,25,26	Com (RS-232-C), Com1, Com2

Reads and counts bytes, from a device, into a buffer in memory.

Function Declaration

```
FUNCTION readbytes (    did       : WORD   ;
                        bytecount : WORD   ;
                        buffer    : ADSMEM ;
                    VAR n_read    : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

FILES

This function reads a specified ("bytecount") number of bytes from a file into memory, and returns in "n_read" the number of bytes actually read.

The count returned in "n_read" is used to determine EOF status for the file. The EOF status is determined when "n_read" is less than "bytecount" (because there are no more bytes to be read).

The input "buffer" is a segmented pointer to the first byte of memory where these bytes will be stored. The returned variable "n_read" is the actual number of bytes read.

RS-232-C

This function transfers a specified number ("bytecount") of bytes from the device buffer specified by "did" to the user specified "buffer".

If the number of characters in the input buffer is less than "bytecount", the driver will wait for the needed characters to arrive.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
17	console
20 - 24	disk files (PCOS)
19,25,26	Com (RS-232-C), Com1, Com2

readline

Reads and counts bytes input from the keyboard, until the first /CR/, into a memory buffer (at a specified address).

Function Declaration

```
FUNCTION readline (    did       : WORD   ;
                       bytecount : WORD   ;
                       buffer    : ADSMEM ;
                   VAR n_read    : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function reads a number of bytes specified in "bytecount" from the standard input device into memory. Input will be terminated either when an input byte is equal to a /CR/ or when the number of input bytes exceeds "bytecount". The /CR/ is not put into the string.

The input DID (which must be 17) identifies standard input. It is the only valid DID for this function. The input "bytecount" specifies the maximum number of bytes to be read, and the input "buffer" is a pointer to the first byte of memory where these bytes will be stored. The output "n_read" is the actual size, in bytes, of the input string.

Characters are echoed to the standard output device and editing features,(/CTRL//H/ i.e.:backspace and /CTRL//I/, i.e.: TAB) and hide mode /CTRL//G/ are implemented. If a /CTRL//C/ is pressed, then the variable "n_read" will be equal to FFFF.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

17	Console (keyboard) only

remove

Removes a specified file name from a disk directory.

Function Declaration

```
FUNCTION remove ( fileadr : ADSMEM ;
                  filelen : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function is used only for disk files. It removes the specified disk file (and related data) from the volume directory.

The input "fileadr" points to the file identifier, and the input "filelen" is the length of the file identifier (complete with an optional volume identifier and/or password).

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Renames a specified file.

Function Declaration

```
FUNCTION rename ( oldfileadr : ADSMEM ;
                  oldflen    : WORD   ;
                  newfileadr : ADSMEM ;
                  newflen    : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function is used only for disks. It will rename the file specified by the old file identifier with the new file name.

The input "oldfileadr" is the pointer to the old file identifier and "oldflen" is the length of the old file identifier (complete with with an optional volume identifier and/or password).

The input "newfileadr" is the pointer to the new file name and "newflen" is the length of this name.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

resetbyte

Resets an input file or device.

Function Declaration

```
FUNCTION resetbyte ( did : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function is used to reset an input device. In the case of the console, it will clear the keyboard ring buffer, and initialize the screen driver. It can also be used with communications (RS-232-C), in which case it re-initializes the hardware and clears the input buffer. The input "DID" identifies the device.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

17	console
19,25,26	Com (RS-232-C), Com1, Com2

Searches the system device table for a device name.

Function Declaration

```
FUNCTION sdevtab (     devname  : ADSMEM ;
                       devlen   : WORD   ;
                   VAR entrynum : BYTE   ;
                   VAR devtype  : BYTE   ;
                   VAR tabptr   : ADSMEM ) : INTEGER ; EXTERN ;
```

This function searches the system device table for the device named. The input parameter "devname" is the address where the first ASCII character of the name is stored, and the input parameter "devlen" is the number of bytes in the name.

If the device name is found the function will return the entry number of the device in the variable "entrynum" and the device type in "devtype" (1 = Read, 2 = Write, 3 = Read/Write). The third returned variable "tabptr" is the address of the first entry in the particular device table.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

search

Searches on a specified disk for a specified file name.

Function Declaration

```
FUNCTION search (     drive      : INTEGER  ;
                      mode       : WORD     ;
                      infileptr  : ADSMEM   ;
                      filelen    : WORD     ;
                      outfileptr : ADSMEM   ;
                  VAR fileptr    : ADSMEM   ;
                  VAR flen       : WORD     ;
                  VAR blocknum   : INTEGER4 ) : INTEGER ; EXTERN ;
```

Characteristics

This function searches on a disk for a file name supplied by the user. The file name may contain wild card characters.

The input parameter "drive" identifies the drive to be searched (input a '-1' for the current drive). The input "mode" is either a '1' for a search from the beginning, or a '0' for a search from the last file found.

The input parameter "infileptr" is an address which points to the file name to be searched for. This file name can contain wild card characters. "filelen" is the length in bytes of the file name. To search for any file input a zero "filelen".

If the specified file name is found it will be written to the memory location pointed to by the input parameter "outfileptr". The address of the file is returned in the returned variable "fileptr", the file length in "flen", while "blocknum" will contain the logical block number of the file descriptor block on the disk.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Writes a word into the Device Parameter Table.

Function Declaration

```
FUNCTION setcontrol ( did     : WORD ;
                      wordnum : WORD ;
                      parword : WORD ) : INTEGER ; EXTERN ;
```

Characteristics

This function allows a single word to be written into the Device Parameter Table (see appendix E). The input "wordnum" is the word number to be written to; the input "parword" is the word to be written to the Device Parameter Table.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

19, 25, 26	Com (RS-232-C), Com1, Com2,

setdate

Sets the system date-clock.

Function Declaration

```
FUNCTION setdate ( dataadr : ADSMEM ;
                   length  : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

The input variable "dataadr" points to an address in the caller's data area which contains the date. The input variable "length" gives the length of the ASCII string.

The format of the data in the string, except for the delimiter, must be:

```
dd:mm:yyyy
```

where "dd" is the day, "mm" is the month, and "yyyy" is the year; leading zeroes need not be supplied.

Any non-numeric character may be used in place of the colon, as shown in the examples for function settime.

The date is initialized to January 1, 1982 at system startup. If only two digits are input for the year, the century is assumed to be 19.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Returns the caller to segmented system mode.

Function Declaration

```
FUNCTION setsysseg : INTEGER ;
```

Characteristics

This function will return the caller to segmented system mode, regardless of which mode the system is in.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

SETTIME

Sets the system clock.

Function Declaration

```
FUNCTION settime ( dataadr : ADSMEM ;
                   length  : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

The input variable "dataadr" points to an address in the caller's data area which contains the time of day. The input variable "length" gives the length of the ASCII string. The format of the data in the string must be:

```
hh:mm:ss
```

where "hh" is the hour (in 24-hour time), "mm" is minutes, and "ss" is seconds. Leading zeros need not be supplied. Any non-numeric character can be selected for delimiter as shown in examples below, using the PCOS SSYS (set system) command.

```
ss 04/15/82,13:12:45

ss "04 15 82",08:10:00
```

Time is initialized to 00:00:00 at system startup. If blanks are selected for delimiters, as in the second example, the expression must be put in quotes.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

setvol

Sets a specified volume for the next access.

Function Declaration

```
FUNCTION setvol ( volnum : WORD ) : INTEGER ;
```

Characteristics

This function sets the volume for the next access. The inputparameter "volnum" is the volume number to be used for the next access.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

stickynew

Allocates a block of bytes from heap that remains allocated after the program executing this function terminates.

Function Declaration

```
FUNCTION stickynew (VAR tableadr : ADSMEM ;
                        size     : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function allocates a block of bytes from the heap, returning in "tableadr" a pointer to the location of the first byte of the block. The block allocated by this function will remain allocated even after the main program terminates.

The input variable "size" is the number of bytes to be allocated.

This function is just like "pnew",with the only difference that the allocated block is not de-allocated when the "calling" program terminates.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Returns the length of the input string.

Function Declaration

```
FUNCTION stringlen (    stringptr : ADSMEM ;
                    VAR strlen    : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

This function returns into the variable "strlen" the length of the input string. The input variable "stringptr" points to the string.

Note that the value returned in "strlen" is either the length read until a null is encountered, or 14, if no null is encountered in that length.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

writebytes

Writes a specified number of bytes from memory to a file or device.

Function Declaration

```
FUNCTION writebytes (    did       : WORD   ;
                         bytecount : WORD   ;
                         buffer    : ADSMEM ;
                     VAR n_write   : WORD   ) : INTEGER ; EXTERN ;
```

Characteristics

FILES

This function writes a specified number ("bytecount") of bytes from memory into a file, and returns in "n_write" the number of bytes actually transferred. Valid DIDs are listed below.

The input "bytecount" is the number of bytes to be transferred. The input "buffer" is a segmented pointer to the first byte in memory from where these bytes will be taken.

The returned "n_write" is the actual number of bytes transfered.

RS-232-C

This function transfers data bytes from the memory buffer specified in "buffer" to the RS-232-C port.

Parameters have the same meaning as for files. If the port is not ready to send, the driver will wait a timeout period after which if nothing is sent the function will return the appropriate error code.

Function Result

The function will return the value zero if and only if there are no errors. If an error occurs then the function will return the error code.

Valid DID Numbers

1 - 15	disk files (BASIC)
17	console
20 - 24	disk files (PCOS)
19,25,26	Com (RS-232-C), Com1, Com2

8. INTRODUCTION TO GRAPHICS

ABOUT THIS CHAPTER

This chapter is an introduction to the graphics facilities available in the M20 Graphics Package. It includes a summary of features and an explanation of graphic concepts; the graphic routines are listed in functional groups. A list of the default conditions for the M20 is given and error reporting is explained.

CONTENTS

INTRODUCTION

The M20 Graphics Library is available in the file "graph.lib", which is an integrated package of over forty routines offering a set of functionalities for two dimensional graphics applications. This library may be called by the PASCAL and Assembly programming languages (for more detail, see chapter 5). The routines in this library can be accessed by the functions and procedures contained in the M20 PASCAL graphics library, "interfg.lib". Throughout this chapter we shall refer to the combined action of these two libraries as the M20 Graphics Package. Chapter 9 contains a detailed description of each routine, in alphabetical order.

SUMMARY OF FEATURES

The M20 Graphics Package presents a consistent and easily comprehensible structure that reflects proposed international standards for such packages.

Besides the full complement of standard output primitives, including lines, polylines, markers, etc., there are several added features:

- line drawings and move operations may be optionally specified as an offset from the current position
- circle, ellipse and rectangle functions are available
- output primitives may be drawn in any of eight colours (on eight-colour systems) or four colours (on four-colour systems)
- polygons, circles and ellipses may be solid filled
- intercharacter spacing for text may be adjusted.

The screen may be subdivided into rectangular regions called view areas. View areas are independent from one another and there may be a maximum of 16 on the screen. If the user tries to draw a picture which does not fit within the view area then only the visible portion of the drawing is displayed and the rest is discarded (clipped).

Pictures, or parts of pictures, may be stored and redrawn when necessary. For every feature that may be set by the M20 Graphics Package, there is an inquiry function which permits the user to request its current state. The inquiry functions return:

- the colour, logic operator and line class for the current view area

- the number of the current view area
- the position and blink rate of the graphics cursor for the current view area
- the location at which graphics output will begin
- the colour number of the pixel which is nearest to a specified point
- the device coordinates of a given point expressed in world coordinates
- the next text entry point and the text cursor blink rate for the current view area
- the size and text parameters of the current view area
- the world coordinate space for the current view area.

The M20 Graphics Package defaults to an operating mode that automatically makes all format decisions (see Default Conditions). The user may change these default conditions to other values which will better suit the specific problem.

CONCEPTS

Graphical output generated by the M20 Graphics Package comprises two general classes of functions: **output primitives** and **primitive attributes.**

Output primitives are abstractions of basic actions that graphics devices can perform, like drawing lines and locating cursors. Output primitives are defined in a two-dimensional user coordinate space (known as world coordinates, see below). The units and the coordinates of the user coordinate space are established by the application program.

Primitive attributes determine the characteristics that an output primitive will possess when displayed on an output device; e.g., line class, colour, intercharacter spacing, etc. Primitive attributes are set modally; i.e., they establish a current value that is assigned to subsequently generated output primitives.

Coordinate data is subjected to transformations that perform a mapping between two coordinate systems, namely:

- **world coordinates,** defined by the user that establish the scaling basis on which the graphical output is described. The world coordinate space definition determines how the coordinates from the application program shall be placed within a view area. When a new view area is created, it will have the same world coordinate space definition as the parent view area, but since the proportions of the two view areas have changed, the shape of subsequent output to those view areas will change too. The world coordinate space defines a view area within the Cartesian plane. DivideViewArea defines a rectangular surface on which the scale of two axes (the x axis and the y axis) is determined. The view area may or may not contain the plane's origin, that is the point of crossing of the two axes at which the coordinates are (0, 0);

- **device coordinates** range from 0 to 511 pixels on the x axis (pixel = picture element, the smallest visible entity on the screen) and from 0 to 255 scanlines on the y axis (scanline = a row of pixels), where each coordinate pair addresses only one specific pixel.

Output primitives and attributes are automatically mapped from the user's world coordinate space to the device coordinates via a transformation which need not concern the user.

The M20 Graphics Package maintains **two current positions,** one for text and one for graphics, and **two cursors,** one for text and one for graphics. Only one of the two cursors (or neither, if so specified) is displayed at any one time. The text current position and the text cursor are always at the same logical location: the point at which the next text output will appear. The graphics current position (the point from which the next graphics output will begin) and the graphics cursor do not coincide. The graphics cursor may be used as an echo symbol to indicate a position on the screen that reflects the values entered by an input device. The current graphics position is used in many but not all graphics output routines, e.g., Polyline will establish its own starting point but moves the current position along as it draws, leaving it at the final point. The circle and ellipse routine (GDP) leaves the current position unaffected.

Some graphics routines use **absolute coordinates,** others use **relative coordinates.** The distinction is that absolute coordinates are distances along the x and y axes from the origin of the Cartesian plane, while relative coordinates are distances along the x and y axes from the current point.

Most of the output routines are affected by the current **colour attributes** and the colour **logic-operator attribute.** There is a foreground colour which determines the colour of text output, and a background colour. There is a graphics colour which determines the colour of graphic output (lines, circles, dots, etc.). These attributes are selected from the range of colours available on the specific M20 configuration.

The colours available on the M20 eight-colour system are: black, red, green, yellow, blue, magenta, cyan and white. The colours available on the M20 four-colour system are four colours chosen from the eight just mentioned. The monochrome system provides two colours, black and white.

The eight colours are numbered from 0 to 7. On four-colour systems, the colours chosen in the range 4..7 map to a value in the range 0..3 via a logical operation. Bits 0 and 2 of the binary representation are OR'd, e.g. the values 4 (100 binary) and 5 (101 binary) give 1 OR 0 = 1 and 1 OR 1 = 1 respectively. This sets the least significant bit (bit 0) and bit 1 remains unchanged. Thus, the values 4 and 5 will become 1 after the logical operation (4 decimal = 100 binary which becomes 01 binary = 1 decimal and 5 decimal = 101 binary which becomes 01 binary = 1 decimal) and the colour is green (if the default value has not been changed). The values 6 and 7 will become 3 after the logical operation (6 decimal = 110 binary which becomes 11 binary = 3 decimal and 7 decimal = 111 binary which becomes 11 binary = 3 decimal) and the colour is red.

The logic-operator attribute determines the resultant output colour, considering the type of graphics routine (text or graphics), the setting of the foreground, background or graphics colour attribute and the colour of the target pixels in the view area. There are six logic operators and each one acts on all pixels in determining what the final colour shall be. The action occurs one pixel at a time, using the colour of the target pixel and that of the new graphics output as operands.

FUNCTIONAL GROUPS

The functional capabilities of the M20 Graphics Package may be divided into four general classes, as follows:

- Transformation and control.

```
ClearViewArea   : clears the specified view area.
CloseGraphics   : closes the graphics session.
CloseViewTrans  : closes the specified view area.
DivideViewArea  : creates a new view area.
Escape          : colours an area.
OpenGraphics    : opens the graphics session.
SelectViewTrans : activates the selected view area.
SetWorldCoordSp : defines the world coordinate space.
```

- Graphics Output.

GDP	: Generalised Drawing Primitive, creates a circle or an ellipse.
GraphCursorAbs	: moves the graphics cursor to a specified absolute position.
GraphCursorRel	: moves the graphics cursor to a specified relative position.
GraphPosAbs	: redefines the current graphics position (absolute).
GraphPosRel	: redefines the current graphics position (relative).
LineAbs	: draws a line from the current graphics position to a specified absolute position.
LineRel	: draws a line from the current graphics position to a specified relative position.
MarkerAbs	: displays a point at a specified absolute position.
MarkerRel	: displays a point at a specified relative position.
PixelArray	: transfers an image onto the screen.
Polyline	: draws a connected sequence of lines.
PolyMarker	: displays the specified points.
TextCursor	: moves the text cursor.

- Graphics Attributes.

SelectCursor	: chooses which cursor, if any, is to be displayed.
SelectGrColour	: selects the colour for subsequent graphics output
SelectTxColour	: selects the colours for subsequent text output.
SetColourLogic	: defines a logic operator that influences the output colour.
SetColourRep	: sets one of the four colour indices to one of the eight M20 colours (four-colour systems only).
SetGrCsrBlnkrate	: sets the blink rate for the graphics cursor.
SetGrCsrShape	: defines the graphics cursor shape.
SetLineClass	: determines the graphics output for the LineAbs, LineRel, and Polyline routines.
SetTextline	: sets the character width and text line height.
SetTxCsrBlnkrate	: sets the blink rate for the text cursor.
SetTxCsrShape	: defines the text cursor shape.

- Inquiry

ErrorInquiry	: returns the error status for the most recently called graphics routine other than the inquiry routines.
InqAttributes	: returns the colour, logic operator and line attributes for the current view area.
InqCurTransNmbr	: returns the number of the current view area.

InqGraphCursor	: returns the position and blink rate of the graphics cursor for the current view area.
InqGraphPos	: returns the location at which new graphics output will begin.
InqPixel	: returns the colour number of the pixel which is nearest to the specified point.
InqPixelArray	: retrieves a rectangular image from the current view area and stores it.
InqPixelCoords	: returns the device coordinates of a given (x,y) pair of world coordinates.
InqTextCursor	: returns the next text entry point and the text cursor blink rate for the current view area.
InqViewArea	: returns the size and text parameters of the current view area.
InqWorldCoordSp	: returns the world coordinate space parameters for the current view area.

ERRORS

Error reporting is handled in two ways. For all routines, an error status is reported; the value zero means no error. If an error has occurred, a value in the range 1 to 255 inclusive is returned. The code numbers used are the standard PCOS error codes with the same meanings (see Appendix D).

For most functions, this status value is transferred to an error status variable maintained by the M20 Graphics Package. There is a function (ErrorInquiry) which returns the current value of this variable (that is, the error status of the most recent Graphics Package routine called other than the inquiry routines).

The inquiry group of functions handles error reports differently. These never touch the error status variable (except for ErrorInquiry which retrieves its value). They report any error directly through an error parameter, and they do not generate an error status.

DEFAULT CONDITIONS

The following is a list of the default conditions that will be assumed unless otherwise specified:

- world coordinates range from 0.0 to 511.0 on the X axis and from 0.0 to 255.0 on the Y axis, coinciding with the device coordinates except that the latter are integer values

- view area number 1 is the full screen, with device coordinates ranging from 0 to 511 on the X axis and from 0 to 255 on the Y axis

- colour depends on the system configuration

  a) the monochrome system sets the background colour to 0 = black and the text and graphics colours to 1 = white

  b) the four-colour system sets the background colour to 0 = black; the text and graphics colours to 1 = green; 2 = blue; 3 = red

  c) the eight-colour system sets 0 = black to the background colour, 1 = green to the text and graphics colour, 2 = blue, 3 = cyan, 4 = red, 5 = yellow, 6 = magenta, 7 = white

- the logic operator is PSET, which displays graphics output in the chosen colour

- the line class is solid line

- no cursor is displayed

- there is one cursor blink per second (one blink includes two changes of state, one from ON to OFF and one from OFF to ON)

- the text cursor shape is 7 pixels wide x 11 scanlines high (within an 8 x 12 space) and is displayed as a rectangle having alternate pixels set

- the graphics cursor shape is 2 pixels wide x 2 scanlines high

- the graphics position is (0.0, 0.0)

- the graphics cursor is at the centre of the screen, i.e. the upper left hand corner of the graphics cursor is at (255, 127)

- there are 16 scanlines per text line and 8 pixels per character

- there are 16 textlines (rows) and 64 characters (columns) per screen.

9. THE M20 PASCAL GRAPHICS LIBRARY

ABOUT THIS CHAPTER

This chapter describes in alphabetical order all the routines provided by the M20 Graphics Package.

CONTENTS

ClearViewArea

Clears the specified view area.

Function Declaration

```
FUNCTION ClearViewArea ( view_area_num : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

view_area-num: specifies the number of the view area to be cleared. It must be within the range 1..16 and must correspond to an existing view area.

Characteristics

This function clears the specified view area, created via DivideViewArea. The view area is not closed, this call merely removes all its current contents. The background colour is unchanged and fills the whole view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

CloseGraphics

Closes the graphics session.

Procedure Declaration

```
PROCEDURE CloseGraphics ; EXTERN ;
```

Characteristics

If this procedure is called it must be the last graphics package call. Calls to graphics routines must be bracketed by the OpenGraphics/CloseGraphics pair, otherwise results are far from guaranteed.

View area definitions and graphics package tables are cleared. The initial default conditions are reset (for these conditions see OpenGraphics).

Errors

This procedure does not return error messages.

Closes the specified view area.

Procedure Declaration

```
PROCEDURE CloseViewTrans ( view_area_num : INTEGER ) ; EXTERN ;
```

Valid Input Values

view_area_num: specifies the number of the view area to be closed. It must be within the range 0..16 and must correspond to an existing view area.

Characteristics

This procedure closes the specified view area created via DivideViewArea. The view area is joined to the one(s) next to it and takes on the same background colour(s) as the adjacent view area(s). The resulting view areas must be rectangular. The enlarged view area(s)' coordinate definitions are adjusted to map the view area's new dimensions.

If the current view area is closed then view area number 1 becomes the current view area. If the parameter "view_area_num" is set to the value zero then all the view areas are closed and view area number 1 becomes the current one, filling the entire screen.

View area number 1 cannot be closed. If the input parameter specifies the value 1, the value of a view area which has not been opened, or a value not within the 0 to 16 range, no error message is generated and the attempt to close the view area has no effect.

Errors

This procedure does not return error messages.

DivideViewArea

Creates a new view area.

Function Declaration

```
FUNCTION DivideViewArea (    div_orient    : INTEGER ;
                             div_point     : INTEGER ;
                         VAR view_area_num : INTEGER ) : INTEGER ;
                                                         EXTERN  ;
```

Valid Input Values

div_orient: 0..3

0: horizontal split; the upper view area is the new one
1: horizontal split; the lower view area is the new one
2: vertical split; the left view area is the new one
3: vertical split; the right view area is the new one.

div_point: defines the division point

- if "div_orient" is 0 or 1 (horizontal split) then this parameter is expressed in scanlines (min = 1, max = current view area height - 1) starting from the top scanline of the current view area

- if "div_orient" is 2 or 3 (vertical split) then this parameter is expressed in number of characters (of 6 or 8 pixels) in the range 1..63 or 1..79, starting from the left side of the current view area. This parameter is always defined in multiples of 8 pixels. If the character width has been set (via SetTextLine) to 6 pixels, the following formula may be used to calculate the actual width of the left view area:

  width_l_v_a=truncate[(num_of_chars*char_width+3)/8]*8

  where "num_of_chars" is equal to the number of characters specified by "div_point" and "char_width" is the current character width in pixels (6 or 8). The remainder before truncation is the right side margin (of the left view area) which is less than 6 pixels wide and so a character can not appear in it. Consequently, the "div_point" parameter may allow one character less than the number it implies;

- if "div_point" is set to the value -1, then the current view area is divided as equally as possible.

Output Values

view_area_num: this parameter returns the new view area's number. The value is within the range 2..16.

Characteristics

This function creates a new view area by splitting the current one as specified by the first two parameters. The third parameter returns the number of the new view area.

The new view area inherits the following attributes from its parent view area: text spacing, colour, and world coordinate space definition.

The initial state is the full screen defined as view area number 1. This can be split into other view areas; adjacent view areas may be closed and joined with it, as long as the resultant view area is rectangular. View area number 1 always exists, it can not be closed. There may be a maximum of 16 view areas at a time. A new view area is assigned the lowest available number in the range 2 to 16 (e.g., if 6 view areas are created and view area number 3 is closed, 3 will be assigned to the view area next created).

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

ErrorInquiry

Returns the error status for the most recently called graphics routine.

Function Declaration

```
FUNCTION ErrorInquiry : INTEGER ; EXTERN ;
```

Output Values

The output value of this function may be:

0: no error for the most recently called graphics routine

1..255: an error has occured in the most recently called graphics routine. The code numbers used are the standard PCOS error codes, with the same meanings.

Characteristics

This function returns the error status for the most recently called graphics routine other than the Inquiry (Inq ...) routines.

The Inquiry class of functions does not alter or test the error status variable: each one has its own error parameter, through which it transmits error messages.

Routines, other than the Inquiry routines, clear the error status variable before execution, and upon completion this variable reflects the error status of the routine. If the value is zero, then no error has occurred.

Errors

This function does not generate errors.

Colours an area.

Function Declaration

```
FUNCTION Escape ( function_num  : INTEGER ;
                  ptr_datastruc : ADSMEM  ) : INTEGER ; EXTERN ;
```

Valid Input Values

function_num: 1

Characteristics

This function paints an area in accordance with the parameters in the data structure pointed to by the input value "ptr_datastruc".

The data structure (e.g., array, record, etc.) must contain the following information:

- an x coordinate (two 16-bit words, single-precision real number, high-order word first)
- a y coordinate (two 16-bit words, single-precision real number, high-order word first)
- two colour numbers, one for painting the area identified by the point (x,y) and one for the border (each colour number is a 16-bit word, integer, high-order first).

The area surrounding the point (x, y) is painted with the colour specified in the data structure, within a contiguous border. No colouring will occur if the point happens to fall on the border. The border must be of only one colour.

The colour numbers have different effects on the monochrome, four-colour and eight-colour systems. However, integers in the range 0..7 will work for both colour parameters without generating errors on all three types of systems. On the monochrome system, 0=black and a value in the range 1..7=white. On four-colour systems, the two colour numbers are indices into a table of four pre-selected colours (see SetColourRep). On eight-colour systems, the values in the range 0..7 have the following meanings: 0=black, 1=green, 2=blue, 3=cyan, 4=red, 5=yellow, 6=magenta, 7=white.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

GDP

Generalised Drawing Primitive, creates a circle or an ellipse.

Function Declaration

```
FUNCTION GDP ( func           : INTEGER ;
               ptr_Xarray     : ADSMEM  ;
               ptr_Yarray     : ADSMEM  ;
               numb_of_points : INTEGER ;
               datarec        : ADSMEM  ) : INTEGER ; EXTERN ;
```

Valid Input Values

func: specifies the geometric function

1: draws a circle
2: draws an ellipse

numb_of_points: specifies the number of points needed for drawing the geometric function

2: for the circle
3: for the ellipse

The two arrays, pointed to by "ptr_Xarray" and "ptr_Yarray" must contain values within the user's world coordinate space definition.

"datarec" is a parameter reserved for future expansion.

Characteristics

The application program must declare and allocate the two coordinate arrays. Each array contains single-precision numbers; the high order word must precede the low order word. The size of each array must be at least large enough to store as many double-word numbers as there are points (2 points for the circle and 3 for the ellipse).

Default values will be assumed for colour, world coordinate space definition, and logic operator.

This function does not affect the current graphics position.

Circle: This function draws a circle if the parameter "func" is set to the value 1.

The world coordinates of the centre point must be stored in the first element of the two arrays (Xarray[1], Yarray[1]). The second element of the two arrays (Xarray[2], Yarray[2]) is the world coordinate of a point on the circumference. The GDP circle function determines the radius by calculating the distance from the centre of the circle to this absolute

location (Xarray[2], Yarray[2]).

The parameter "numb_of_points" is set to the value 2 and indicates that the arrays specify two geometric points.

The output generated by this function is always a circle, regardless of the coordinate space definition.

If the coordinates generate a circle larger than the view area then the portions that lie outside the view area are clipped.

Ellipse: This function draws an ellipse (parallel to the x or y axis) if the parameter "func" is set to the value 2.

The world coordinates of the centre point must be stored in the first element of the two arrays (Xarray[1], Yarray[1]). The second and third elements (Xarray[2], Yarray[2]) and (Xarray[3], Yarray[3]) contain the world coordinates of one end of the minor axis (either one will do), and of one end of the major axis. (It does not matter which axis point comes first.)

The parameter "numb_of_points" is set to the value 3 and indicates that the arrays specify three geometric points.

If the coordinates generate an ellipse larger than the view area then the portions that lie outside the view area are clipped.

The exact shape of the ellipse may vary depending on the coordinate space definition.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Moves the graphics cursor to the specified absolute position.

Function Declaration

```
FUNCTION GraphCursorAbs ( x : REAL4 ;
                          y : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

The two values "x" and "y" (single-precision real numbers) must specify a point within the user's world coordinate space definition.

Characteristics

This function moves the graphics cursor to the absolute position specified in world coordinates. The graphics cursor is displayed only if the SelectCursor function has been previously invoked, setting the "cursor_num" parameter to the value 1.

If the coordinates specify a point which is outside the current view area then the current position of the graphics cursor remains unchanged and an error code is generated.

The current graphics position is not associated with the position of the graphics cursor. The position of the graphics cursor coincides with that of the current graphics position only when both are assigned the same coordinates. This separation allows the application program to use the graphics cursor as an echo symbol to indicate a position on the screen that reflects the values entered by an input device.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

GraphCursorRel

Moves the graphics cursor to a specified relative position.

Function Declaration

```
FUNCTION GraphCursorRel ( dx : REAL4 ;
                          dy : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

The resulting position must fall within the user's world coordinate space definition. "dx" and "dy" must be single-precision real numbers.

Characteristics

This function moves the graphics cursor to a new position which is obtained by adding the input values dx,dy (which specify the distance between the old position and the new one in world coordinates) to the old graphics cursor position.

The graphics cursor is displayed only if the SelectCursor function has been previously invoked, setting the "cursor_num" parameter to the value 1.

If the resulting position is outside the current view area then the current position of the graphics cursor remains unchanged and an error code is generated.

The current graphics position is not associated with the position of the graphics cursor. The position of the graphics cursor coincides with that of the current graphics position only when both are assigned the same coordinates. This separation allows the application program to use the graphics cursor as an echo symbol to indicate a position on the screen that reflects the values entered by an input device.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Redefines the current graphics position (absolute).

Function Declaration

```
FUNCTION GraphPosAbs ( x : REAL4 ;
                       y : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

The two values "x" and "y" must be single-precision real numbers.

Characteristics

This function redefines the current graphics position for subsequent graphics output. The input values specify an absolute location in world coordinates. Any subsequent graphics output that uses the current graphics position as a starting point will use this redefined position.

The current graphics position is not associated with the position of the graphics cursor. The position of the graphics cursor coincides with that of the current graphics position only when both are assigned the same coordinates. This separation allows the application program to use the graphics cursor as an echo symbol to indicate a position on the screen that reflects the values entered by an input device.

The specified point becomes the current graphics position even if it is not within the current view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

GraphPosRel

Redefines the current graphics position (relative).

Function Declaration

```
FUNCTION GraphPosRel ( dx : REAL4 ;
                       dy : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

"dx" and "dy" must be single-precision real numbers.

Characteristics

This function redefines the current graphics position for subsequent graphics output. The new graphics position is obtained by adding the input values dx, dy (which specify a distance in world coordinates) to the previous graphics position. The next graphics output that uses the current graphics position as a starting point will use this redefined position.

The current graphics position is not associated with the position of the graphics cursor. The position of the graphics cursor coincides with that of the current graphics position only when both are assigned the same coordinate point. This separation allows the application program to use the graphics cursor as an echo symbol to indicate a position on the screen that reflects the values entered by an input device.

The specified point becomes the current graphics position even if it is not within the current view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

InqAttributes

Returns the colour, logic operator and line attributes for the current view area.

Function Declaration

```
FUNCTION InqAttributes ( VAR graphics_col : INTEGER ;
                         VAR foregd_col   : INTEGER ;
                         VAR backgd_col   : INTEGER ;
                         VAR logic_oper   : INTEGER ;
                         VAR lineclass    : INTEGER ) : INTEGER ;
                                                        EXTERN  ;
```

Output Values

graphics_col: 0..7

foregd_col : 0..7

backgd_col : 0..7

logic_oper : 0..5

lineclass : 0..2

Characteristics

This function returns the following information for the current view area:

- "graphics_col": current graphics colour (see SelectGrColour)
- "foregd_col": text foreground colour (see SelectTxColour)
- "backgd_col": background colour (see SelectTxColour and ClearViewArea)
- "logic_oper": logic operator for colour (see SetColourLogic)
- "lineclass": line class (see SetLineClass).

If the view area is undefined (see DivideViewArea) an error code is returned.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

InqCurTransNumber

Returns the number of the current view area.

Function Declaration

```
FUNCTION InqCurTransNumber ( VAR view_area_num : INTEGER ) : INTEGER ;
                                                             EXTERN  ;
```

Output Values

view_area_num: 1..16

Characteristics

This function returns the identification number of the current view area. This number may be used for:

- selecting a different view area
- redefining the view area's world coordinate space
- clearing the view area's contents
- closing the view area
- retrieving information about the view area (e.g., colour, coordinates).

Errors

The only value returned by this function is 0, no errors.

InqGraphCursor

Returns the position and blink rate of the graphics cursor for the current view area.

Function Declaration

```
FUNCTION InqGraphCursor ( VAR x          : REAL4   ;
                          VAR y          : REAL4   ;
                          VAR blink_rate : INTEGER ) : INTEGER ;
                                                       EXTERN  ;
```

Output Values

x : x world coordinate of the graphics cursor

y : y world coordinate of the graphics cursor

blink_rate : 0..20

Characteristics

This function returns the location (x,y) in world coordinates of the graphics cursor and its blink rate expressed in state changes per second (from OFF to ON or from ON to OFF), rounded to the nearest 50 milliseconds. See SetGrCsrBlinkrate.

The graphics cursor is placed with its upper left hand corner of its 8x12 pixel shape at this (x,y) position.

The graphics cursor position and the graphics position are generally not the same; the graphics cursor merely marks a position within the view area.

The graphics cursor position and the text cursor position are entirely independent of each other. Only one of these two cursors (or neither, if so specified) appears at any one time.

Errors

The only value returned by this function is 0, no errors.

InqGraphPos

Returns the location at which new graphics output will begin.

Function Declaration

```
FUNCTION InqGraphPos ( VAR x : REAL4 ;
                       VAR y : REAL4 ) : INTEGER ; EXTERN ;
```

Output Values

x : x world coordinate at which the next graphics output will begin

y : y world coordinate at which the next graphics output will begin.

Characteristics

This function returns the location (x,y), in world coordinates, within the current view area at which new graphics output will begin (e.g., a LineRel call would generate a line with the first end at this point).

Errors

The only value returned by this function is 0, no errors.

InqPixel

Returns the colour number of the pixel which is nearest to the specified point.

Function Declaration

```
FUNCTION InqPixel (    X_world_coord : REAL4    ;
                       Y_world_coord : REAL4    ;
                   VAR colour_num    : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

X_world_coord: x coordinate within the user's world coordinate space definition; single-precision real

Y_world_coord: y coordinate within the user's world coordinate space definition; single-precision real

Output Values

colour_num: 0..7

On monochrome systems:

"colour_num"	colour
0	black
1	white

On four-colour systems, "colour_num" returns a value in the range 0..3 (see SetColourRep). The default values are:

"colour_num"	colour
0	black
1	green
2	blue
3	red

On eight-colour systems:

"colour_num"	colour
0	black
1	green
2	blue
3	cyan
4	red
5	yellow
6	magenta
7	white

Characterisitcs

This function returns the colour number of the pixel which is nearest to the specified (world coordinate) point in the current view area.

In the monochrome and eight-colour systems, the colour numbers are predefined; in four-colour systems, the value is an index to a table (see SetColourRep) of pre-selected colours (four colours selected from the eight available).

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Retrieves a rectangular image from the current view area and stores it.

Function Declaration

```
FUNCTION InqPixelArray (    X_width      : REAL4  ;
                            Y_height     : REAL4  ;
                            ptr_array    : ADSMEM ;
                       VAR invalid_code  : INTEGER) : INTEGER ; EXTERN ;
```

Valid Input Values

X_width: width of the rectangle to be retrieved, expressed in world coordinates, single precision real

Y_height: height of the rectangle to be retrieved, expressed in world coordinates, single precision real.

Output Values

invalid_code: this parameter reports discovery of invalid pixel colour values if it is set to 1; if all pixel values are valid this parameter is set to 0.

Characteristics

This function retrieves a rectangular image from the current view area, and stores it in the array pointed to by "ptr_array" to be displayed later. The inverse function is accomplished by PixelArray.

The upper left-hand corner of the rectangle to be retrieved from the screen is placed at the current graphics position.

The two input parameters "X_width" and "Y_height" specify the rectangle's dimensions in world coordinates. These dimensions are transformed into device coordinates (pixels). The size of the storage array depends on the total number of pixels in the rectangle. The user may calculate this total number of pixels by:

1. retrieving the device coordinates for each corner of the rectangle (via InqPixelCoords) only if the default world coordinates have been changed

2. calculating the rectangle's width and height in pixels

3. applying the "array_size" formula (see below).

The application program is responsible for knowing the required array size and allocating space for it.

The array contains the bit images of the scanlines within the rectangle, packed 16 bits per array entry. Each scanline image will begin with the first bit of the scanline in bit 15 of the first array entry (left justified). The size of the array (in words) may be calculated according to the following formula:

array_size=truncate[(pixel_width+15)/16]*pixel_height*colour_planes+3

where "colour_planes" is the number of colour planes in the system configuration. Each colour plane provides one bit (i.e. two states) per pixel. With two colour planes, each pixel is represented by two bits and thus four states are possible. By extension, three colour planes provide eight states. Therefore, monochrome has 1 colour plane, four-colour has 2 colour planes and eight-colour has 3 colour planes.

The extra 3 words in the array (at the beginning) contain the rectangle's width, height and special codes related to the conditions in which the array was created (number of colour planes, etc.). The array is one dimensional. The maximum size of the array is that needed for a full screen rectangle, assuming that there is sufficient memory in the system configuration for storing an array that large.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Returns the device coordinates (expressed in pixels) of a given point expressed in world coordinates.

Function Declaration

```
FUNCTION InqPixelCoords (     X_world_coord  : REAL4   ;
                              Y_world_coord  : REAL4   ;
                          VAR X_device_coord : INTEGER ;
                          VAR Y_device_coord : INTEGER ) : INTEGER ;
                                                           EXTERN  ;
```

Valid Input Values

X_world_coord: X-coordinate within the user's world coordinate space definition; single-precision real number

Y_world_coord: Y-coordinate within the user's world coordinate space definition; single-precision real number

Output Values

X_device_coord: 0..511

Y_device_coord: 0..255

Characterisitcs

This function returns the device coordinates, expressed in pixels, for the input world coordinates. The device coordinates are calculated with respect to the borders of the current view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

InqTextCursor

Returns the next text entry point and the text cursor blink rate for the current view area.

Function Declaration

```
FUNCTION InqTextCursor ( VAR column     : INTEGER ;
                         VAR row        : INTEGER ;
                         VAR blink_rate : INTEGER ) : INTEGER ; EXTERN ;
```

Output Values

column: 1..64 or 1..80

row: 1..16 or 1..25

blink_rate: 0..20

Characterisitcs

This function returns the next text entry point, which coincides with the location of the text cursor, and the text cursor blink rate for the current view area. See SetTxCsrBlinkrate.

The text cursor position is given in number of columns (e.g., number of characters) from the view area's left edge and of rows (e.g., number of text lines) from the view area's top edge.

The cursor blink rate is expressed in state changes per second (from OFF to ON or from ON to OFF) rounded to the nearest 50 milliseconds.

If the information is not available it is because the view area is too small to contain text. An error code is returned and the other output parameters remain undefined.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

InqViewArea

Returns the size and text parameters of the current view area.

Function Declaration

```
FUNCTION InqViewArea ( VAR view_area_width  : INTEGER ;
                       VAR view_area_height : INTEGER ;
                       VAR text_char_width  : INTEGER ;
                       VAR text_line_height : INTEGER ) : INTEGER ;
                                                          EXTERN  ;
```

Output Values

view_area_width : 1..64 bytes

view_area_height : 1..256 scanlines

text_char_width : 6 or 8 pixels

text_line_height : 10..16 scanlines

Characteristics

This function returns the current view area's width (in bytes) and height (in scanlines), and the current character's width (in pixels) and height (in scanlines).

Errors

The only value returned by this function is 0, no errors.

InqWorldCoordSpace

Returns the world coordinate space parameters for the current view area.

Function Declaration

```
FUNCTION InqWorldCoordSpace ( VAR x0 : REAL4 ;
                              VAR y0 : REAL4 ;
                              VAR x1 : REAL4 ;
                              VAR y1 : REAL4 ) : INTEGER ; EXTERN ;
```

Output Values

x0: x coordinate, within the user's world coordinate space definition, of the lower left-hand corner of the current view area

y0: y coordinate, within the user's world coordinate space definition, of the lower left-hand corner of the current view area

x1: x coordinate, within the user's world coordinate space definition, of the upper right-hand corner of the current view area

y1: y coordinate, within the user's world coordinate space definition, of the upper right-hand corner of the current view area.

Characteristics

This function returns the world coordinates of the lower left-hand corner (x0, y0) and of the upper right-hand corner (x1, y1) respectively, of the current view area.

These coordinates do not determine the proportions of the view area; they determine how points in world coordinates will map to the current view area.

Errors

The only value returned by this function is 0, no errors.

Draws a line from the current graphics position to the specified absolute position.

Function Declaration

```
FUNCTION LineAbs ( x : REAL4 ;
                   y : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

The two values "x" and "y" must be single-precision real numbers.

Characteristics

This function draws a line from the current graphics position to the absolute (x,y) position which is specified in world coordinates.

Default values will be assumed for coordinate space, colour, logic operator, and line class.

If the (x,y) coordinates specify a point which is outside the view area but within the range of a single-precision floating-point number, then a line is drawn in the direction of the specified point but is clipped on the view area boundary.

The specified point becomes the current graphics position, even if it is outside the view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

LineRel

Draws a line from the current graphics position to a specified relative position.

Function Declaration

```
FUNCTION LineRel ( dx : REAL4 ;
                   dy : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

"dx" and "dy" must be single-precision real numbers.

Characteristics

This function draws a line, the length and direction of which are specified in world coordinates by the dx and dy input parameters, starting from the current graphics position.

Default values will be assumed for coordinate space, colour, logic operator, and line class.

If the point, resulting from the input distances, is outside the view area but within the range of a single-precision floating-point number, then a line is drawn in the specified direction but is clipped on the view area boundary.

The resulting point becomes the current graphic position, even if it is outside the view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Displays a point at the specified absolute position.

Function Declaration

```
FUNCTION MarkerAbs ( x : REAL4 ;
                     y : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

The two values "x" and "y" must be single-precision real numbers.

Characteristics

This function displays a point at the absolute (x,y) position which is specified in world coordinates.

Default values will be assumed for coordinate space, colour, and logic operator.

If the (x,y) coordinates specify a point which is outside the view area but within the range of a single-precision floating-point number, then no point is displayed. The specified point becomes the current graphics position, even if it is outside the view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

MarkerRel

Displays a point at a specified distance from the current graphics position.

Function Declaration

```
FUNCTION MarkerRel ( dx : REAL4 ;
                     dy : REAL4 ) : INTEGER ; EXTERN ;
```

Valid Input Values

"dx" and "dy" must be single-precision real numbers.

Characteristics

This function displays a point at a specified (dx,dy) distance from the current graphics position. The distance is specified in world coordinates.

Default values will be assumed for coordinate space, colour, and logic operator.

If the point, resulting from the input distances, is outside the view area but within the range of a single-precision floating-point number, then the point is not displayed. The resulting point becomes the current graphics position.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

OpenGraphics

Sets up the M20 for creating graphics.

Procedure Declaration

```
PROCEDURE OpenGraphics ; EXTERN ;
```

Characteristics

This procedure initialises the graphics environment and thus must be the first graphics call within an application program. It sets default conditions as follows:

- a single view area, labelled 1
- world coordinates coincide with device coordinates (0.0-511.0 pixels x 0.0-255.0 scanlines)
- black as the background colour
- white as the foreground and text colours for the black and white system and green for the colour system
- no cursor displayed
- a blank screen.

It may also be used to reinitialise the graphics environment, thus clearing the effects of all the preceding graphics calls. The application program handles subsequent graphics functions and procedures and their output as if starting afresh.

Errors

This procedure does not return error messages.

PixelArray

Transfers an image onto the screen.

Function Declaration

```
FUNCTION PixelArray ( x_width   : REAL4  ;
                      y_height  : REAL4  ;
                      ptr_array : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function retrieves a rectangular image stored into a one-dimensional array (pointed to by "ptr_array") and displays it on the screen. The rectangular image stored in memory is part of (or all) a picture previously displayed on a view area.

The size of the rectangular image to be displayed is "x_width" wide and "y_height" high. These two values are in world coordinates.

The image is displayed with the rectangle's upper left hand corner at the current graphics position. The default value for logic operator is assumed.

The two parameters "x_width" and "y_height" need not correspond to the full size of the image implied by the array. If the rectangular image stored in the array is relatively large compared to this function's arguments "x_width" and "y_height", then only part of the stored image is displayed. The right and bottom edge of the image are clipped.

If the rectangular image stored in the array is smaller than that implied by the arguments "x_width" and "y_height" of this function, then the full picture will appear. This will not extend to the right and bottom borders implied by the two arguments.

If the current graphics position is too close to the right and/or bottom edge of the screen for the entire image to be displayed, then only part of the image is displayed and the rest of it is clipped at the screen edge.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Polyline

Draws a connected sequence of lines.

Function Declaration

```
FUNCTION Polyline ( numb_of_points : INTEGER ;
                    ptr_Xarray     : ADSMEM  ;
                    ptr-Yarray     : ADSMEM  ) : INTEGER ; EXTERN ;
```

Valid Input Values

numb_of_points: this value must be equal to or greater than 2.

The two arrays, pointed to by "ptr_Xarray" and "ptr_Yarray", must contain single-precision real numbers.

Characteristics

This function draws lines connecting the points specified by the two arrays. The two arrays are the same size and contain single-precision real numbers. A coordinate is made up of element Xarray[J] of the first array and element Yarray[J] of the second array. The parameter "numb_of_points" contains an integer specifying the number of points to be connected. The points are absolute locations in the world coordinates.

Default values will be assumed for coordinate space, colour, logic operator, and line class.

The application program must declare and allocate the two coordinate arrays. Each array contains single-precision real numbers; the high order word must precede the low-order word. The size of each array must be at least large enough to store as many double-word numbers as there are points.

The figure will not be a closed polygon unless the first and last points specified by the arrays coincide.

If the coordinates specify points that are not within the view area, then the figure will be clipped on the view area boundary. If the last point is outside the view area, it nevertheless becomes the current graphics position.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

PolyMarker

Displays the specified points.

Function Declaration

```
FUNCTION PolyMarker ( num_points : INTEGER ;
                      ptr_Xarray : ADSMEM  ;
                      ptr_Yarray : ADSMEM  ) : INTEGER ; EXTERN ;
```

Valid Input Values

num_points: this value must be equal to or greater than 1.

The two arrays, pointed to by ptr_Xarray and ptr_Yarray must contain single-precision real numbers.

Characteristics

This function displays the number of points specified by "num_points"; each one is identified by the coordinates specified by the two arrays. A coordinate is made up of element Xarray[J] of the first array and element Yarray[J] of the second array. The two arrays are the same size and contain single-precision real numbers. The points are absolute locations in world coordinates.

Default values will be assumed for coordinate space, colour, and logic operator.

The two arrays, pointed to by "ptr_Xarray" and "ptr_Yarray", must contain values within the user's world coordinate space definition. The application program must declare and allocate the two coordinate arrays. Each array contains single-precision real numbers; the high order word must precede the low order word. The size of each array must be at least large enough to store as many double-word numbers as there are points.

The coordinates which specify points that are outside the view area will not be displayed and no error message is generated. However, the current graphics position will track these non-visible points and if the last point is outside the view area it nevertheless becomes the current graphics position.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SelectCursor

Chooses which cursor is to be displayed.

Function Declaration

```
FUNCTION SelectCursor ( cursor_num : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

cursor_num : 0..2 ; specifies the cursor to be displayed

0: neither cursor
1: the graphics cursor
2: the text cursor.

Characteristics

This function chooses which cursor (if any) is to be displayed.

If selected, the text cursor is displayed and text will be displayed starting from that position.

If selected, the graphics cursor is displayed with its upper left hand corner at the current graphics position. However, subsequent graphics output will not start from this point unless the current graphics position has been updated to this same location.

The text and graphics cursor do not usually occupy the same position. The two cursors cannot be displayed simultaneously.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SelectGrColour

Selects the colour for subsequent graphics output.

Function Declaration

```
FUNCTION SelectGrColour ( colour_code : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

On monochrome systems, "colour_code" selects either black or white to be the graphics colour attribute:

"colour_code"	graphics colour attribute
0	black
1..7	white

On four-colour systems, "colour_code" selects the colour attribute indirectly by acting as an index into a table of four colours preselected from the eight possible colours (see SetColourRep):

"colour_code"	graphics colour attribute
0..3	the colour attribute associated with each one of the four values 0, 1, 2 and 3, depends on the values set by default or via SetColourRep.
4..7	The values in this range map to a value in the range 0..3 via a logical operation (see the following note).

Note: Bits 0 and 2 of the binary representation are OR'd, e.g., the values 4 (100 binary) and 5 (101) give 1 OR 0 = 1 and 1 OR 1 = 1 respectively. This sets the least significant bit (bit 0) and bit 1 remains unchanged. Thus, the values 4 and 5 will become 1 after the logical operation (4 decimal = 100 binary which becomes 01 binary = 1 decimal and 5 decimal = 101 binary which becomes 01 binary = 1 decimal) and the colour is green (if the default value has not been changed). The values 6 and 7 will become 3 after the logical operation (6 decimal = 110 binary which becomes 11 binary = 3 decimal and 7 decimal = 111 binary which becomes 11 binary = 3 decimal) and the colour is red.

On eight-colour systems, "colour-code" selects the colour attribute directly, according to the following table:

"colour_code"	graphics colour attribute
0	black
1	green
2	blue
3	cyan
4	red
5	yellow
6	magenta
7	white

Characteristics

This function selects the specified colour for subsequent graphics output. There are different effects on monochrome, four-colour and eight-colour systems.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SelectTxColour

Selects the colours for subsequent text output.

Function Declaration

```
FUNCTION SelectTxColour ( fg_code : INTEGER ;
                          bg_code : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

fg_code: 0..7

bg_code: 0..7

On monochrome systems, if the foreground colour "fg_code" is set to black (0), the background colour "bg_code" may be set to any value in the range 1..7 (white). The default value for the background colour is black.

On four colour systems, each parameter selects the colour attribute indirectly by acting as an index into a table of four colours preselected from the eight possible colours (see SetColourRep):

"bg_code" and "fg_code"	text colour attribute
0..3	the colour attribute, associated with each one of the four values 0, 1, 2 and 3, depends on the values set via SetColourRep.
4..7	the colours selected are not easily predictable.

On eight colour systems, each parameters selects the colour attribute directly, according to the following table:

"fg_code" and "bg_code"	text colour attribute
0	black
1	green
2	blue
3	cyan
4	red
5	yellow
6	magenta
7	white

Characteristics

This function specifies the colours to be used as the foreground and background of text output. Text is displayed in the foreground colour.

The background colour also affects the ClearViewArea function and the "PRESET" logic-operator (see SetColourLogic).

The values set by this function hold until it is called again.

There are different effects on monochrome, four-colour and eight-colour systems.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SelectViewTrans

Activates the selected view area.

Function Declaration

```
FUNCTION SelectViewTrans ( view_area_num : INTEGER ) : INTEGER ;
                                                       EXTERN  ;
```

Valid Input Values

view_area_num : 1..16

Characteristics

This function activates the specified view area which has previously been defined via DivideViewArea. All text and graphics output will be displayed on this view area and is entered in accordance with its attributes (colour, world coordinate space, text spacing, current text and graphics positions, etc.).

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetColourLogic

Defines a logic operator that influences the output colour.

Function Declaration

```
FUNCTION SetColourLogic ( logop_code : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

logop_code: 0..5

0 PSET: graphics output is displayed in the default colour or in the colour specified via the last SelectGrColour call.

1 XOR: the graphics colour and the colour of the target pixel are logically XOR'd. Graphics output is drawn in the resulting colour, e.g., if the current graphics colour is blue (010 binary) and the pixels on which the geometrical output will be drawn are yellow (101 binary), then the resulting colour is white (010 XOR 101 = 111).

2 AND: the graphics colour and the colour of the target pixel are logically AND'd. Graphics output is drawn in the resulting colour, e.g., if the current graphics colour is blue (010 binary) and the pixels on which the geometrical output will be drawn are yellow (101 binary), then the resulting colour is black (010 AND 101 = 000).

3 NOT: this is a unary operator that complements the colour of the target pixel (the current graphics colour is irrelevant), e.g., if the target pixel is yellow (101 binary) then the resulting colour is blue (010 binary).

4 OR: the graphics colour and the colour of the target pixel are logically OR'd. Graphics output is drawn in the resulting colour, e.g., if the current graphics colour is blue (010 binary) and the pixels on which the geometrical output will be drawn are yellow (101 binary) then the resulting colour is white (010 OR 101 = 111).

5 PRESET: the graphics colour is set to the background colour which is black on the monochrome and eight colour systems if the default values remain unchanged; it is also black on the four colour system if the default values remain unchanged.

Characteristics

This function specifies a logic operator that will influence the output colour (for all subsequent output except text) on a pixel-by-pixel basis.

When new output is displayed the logic operation is applied one pixel at a time. The logic operations deal with the numbers in the 0..7 range, as three-bit binary quantities.

The specific results vary depending on the system configuration. Monochrome systems transform the numbers in the range 2..7 to 1, thus the only operands are 0 and 1, which are the colours black and white respectively.

Eight colour systems make no transformation and deal with the numbers directly as colours, with corresponding results.

Four colour systems treat the numbers not directly as colours but as indices into the four-colour table. Predicting the final result is possible but requires some calculation.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Sets one of the four colour indices to one of the eight M20 colours.

Function Declaration

```
FUNCTION SetColourRep ( index  : INTEGER ;
                        colour : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

index 0..3

colour: 0..7; the following table shows the corresponding colour attributes:

"colour"	graphics colour attribute
0	black
1	green
2	blue
3	cyan
4	red
5	yellow
6	magenta
7	white

Characteristics

This function is used on four-colour systems and has no effect on monochrome and on eight-colour systems. It sets one of the four colour indices to one of the eight M20 colours.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetGrCsrBlinkrate

Sets the blink rate for the graphics cursor.

Function Declaration

```
FUNCTION SetGrCsrBlinkrate ( rate : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

rate: 0..20

```
0:      the cursor is left ON continuously
1..20:  the cursor blinks "rate"/2 times per second
```

Characteristics

This function sets the blink rate for the graphics cursor, from the steady state to the specified state changes per second (from OFF to ON or from ON to OFF). The specified value is truncated to the nearest 50 milliseconds.

This function does not select which cursor is to be displayed, the SelectCursor function does this.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Defines the graphics cursor shape.

Function Declaration

```
FUNCTION SetGrCsrShape ( ptr_array : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function defines the graphics cursor shape according to the contents of the array pointed to by "ptr_array". This array consists of 6 one-word (2 bytes) elements, each containing a 16-bit unsigned integer. Each byte is a bit-map of a scanline of the cursor. The first element's high-order byte is the top scanline of the new cursor; the sixth's element low-order byte is the last scanline of the new cursor.

This function does not select which cursor is to be displayed, the SelectCursor function does this.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetLineClass

Determines the graphics output for the LineAbs and LineRel functions.

Function Declaration

```
FUNCTION SetLineClass ( class_num : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

class_num: 0..2

0: line
1: hollow rectangle
2: solid rectangle.

Characteristics

This function determines whether the graphics output for the LineAbs and LineRel functions will be a line, a hollow rectangle or a solid rectangle. In the latter two cases, the world coordinates specified in the LineAbs and LineRel functions constitute the end points of the diagonal of the rectangle.

The graphics output will be displayed in the current graphics colour.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetTextline

Sets the character width and text line height.

Function Declaration

```
FUNCTION SetTextline ( chr_width      : INTEGER ;
                       textline_height : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

chr_width: 6 or 8

textline_height: 10..16

Characteristics

This function sets the width (in pixels) and the text line height (in scanlines) of the character space. It is the space around each character which grows or shrinks, the individual character size remaining unchanged.

The values set by this function hold for the current view area and all subdivisions of it, or until the function is called again.

This setting influences the width of subsequent view area definitions. In fact, DivideViewArea's second parameter "div_point" establishes the division point of the view area. For vertical divisions, "div_point" expresses the number of characters from the old view area's left edge. If the character width is of 6 pixels rather than 8, then the left view area will be smaller than it would have been with 8 pixel characters.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetTxCsrBlinkrate

Sets the blink rate for the text cursor.

Function Declaration

```
FUNCTION SetTxCsrBlinkrate ( rate : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

rate: 0..20

```
0:      the cursor is left ON continuoulsy
1..20:  the cursor blinks "rate"/2 times per second
```

Characteristics

This function sets the blink rate for the text cursor, from the steady state to the specified state changes per second (from OFF to ON or from ON to OFF), e.g., if "rate" is set to the value 8, then there are 8 states per second, 4 "on" states and 4 "off" states. The specified value is truncated to the nearest 50 milliseconds.

This function does not select which cursor is to be displayed, the SelectCursor function does this.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Defines the text cursor shape.

Function Declaration

```
FUNCTION SetTxCsrShape ( ptr_array : ADSMEM ) : INTEGER ; EXTERN ;
```

Characteristics

This function defines the text cursor shape according to the contents of the array pointed to by "ptr_array". This array consists of 6 one-word (2 bytes) elements, each containing a 16-bit unsigned integer. Each byte is a bit-map of a scanline of the cursor. The first element's high-order byte is the top scanline of the new cursor; the sixth element's low-order byte is the last scanline of the new cursor.

If the most significant bit of each byte is set then the leftmost column of pixels will touch the character preceding it.

This function does not select which cursor is to be displayed, the SelectCursor function does this.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

SetWorldCoordSpace

Defines the world coordinate space.

Function Declaration

```
FUNCTION SetWorldCoordSpace ( view_area_num  : INTEGER ;
                              x0, y0, x1, y1 : REAL4    ) : INTEGER ;
                                                            EXTERN  ;
```

Valid Input Values

view_area_num: 1..16.

The coordinates x0, y0, x1, y1 are single-precision real numbers.

Characteristics

This function defines the user coordinate space, known as the world coordinate space. The function may be called again to redefine the user's world coordinate space when required.

The input coordinates determine the scaling interpretation within the specified view area and not the view area's size which is determined via the DivideViewArea function.

All subsequent graphics coordinates within the view area will be scaled by a transformation routine using the input coordinates (x0, y0) and (x1, y1), which define the endpoints of a diagonal of the entire view area. (x0, y0) are the coordinates of the lower left-hand corner and (x1, y1) are those of the upper right-hand corner of the view area.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions. If there are no errors, a zero is returned.

Moves the text cursor.

Function Declaration

```
FUNCTION TextCursor ( column : INTEGER ;
                      row    : INTEGER ) : INTEGER ; EXTERN ;
```

Valid Input Values

column: 1..64 or 1..80

row: 1..16 or 1..25

Characteristics

This function moves the text cursor and thereby determines the next screen position at which text will be displayed in the current view area. The text cursor is displayed only if the SelectCursor function has been previously invoked, setting "cursor_num" to the value 2.

The parameter "column" contains a number in the range 1..64 or 1..80, depending on whether the character's width is 8 or 6 pixels respectively. The parameter "row" contains a number in the range 1..16 or 1..25, depending on the character's height (see SetTextline).

A full-screen view area may be 64 columns wide and 16 rows high or 80 columns wide and 25 rows high. The dimension which is current determines the position of anything specified in terms of character counts. If the current view area is smaller than full-screen then the amount of text that it may contain depends on the dimensions of the view area.

If the coordinates specify a point which is outside the current view area then the current position of the text cursor is unchanged.

Errors

If there are any errors, the status code is returned by this function. The code numbers correspond to the standard PCOS error codes, with the same meanings. See APPENDIX D for the error descriptions.
If there are no errors, a zero is returned.

A. IMPLEMENTATION CHARACTERISTICS

ABOUT THIS APPENDIX

This appendix describes the Olivetti implementation of the MS-Pascal language for the M20 operating system (PCOS).

CONTENTS

INTRODUCTION

Microsoft Pascal has been implemented for a number of different microcomputer operating systems. This appendix describes the Olivetti implementation of the MS-Pascal language for the M20 operating system (PCOS). It discusses additions and restrictions to the language described in the "PASCAL Language Reference Manual" and identifies features of MS-Pascal that are not yet implemented.

IMPLEMENTATION ADDITIONS

The following additions have been made to the language described in the Reference Manual.

1. Program parameters are available. When a program starts, there is a prompt for every program parameter. You may also give program parameters on the command line with which you invoke the program. If a program requires more parameters than appear on the command line, the remaining parameters are prompted for.

   For example, if you compile and LINK the following program:

```
PROGRAM DEMO (INFILE, OUTFILE, P1, P2, P3);
VAR  INFILE, OUTFILE : TEXT;
     P1, P2, P3   : INTEGER;
BEGIN
  .
  .
END.
```

   into a load file "demo.cmd", then from the command line, you could run this program as follows:

```
de 1:DATA1,1:DATA2,7,8,123 /CR/
```

   If you give only the first parameter on the command line, the program

will proceed to prompt you as shown below:

```
de 1:DATA1 /CR/

OUTFILE: 1:DATA2,7 /CR/
P2: 8 /CR/
P3: 123 /CR/
```

An LSTRING parameter value of NULL cannot be read from the command line and is assumed to be missing. You can enter it by pressing the RETURN key in response to the prompt.

2. Bankers' rounding is used when ROUNDing real numbers that end with .5; that is, odd numbers are rounded up to an even integer, even numbers are rounded down to an even integer. For example:

```
ROUND(4.5) = 4

ROUND (207.5) = 208
```

IMPLEMENTATION RESTRICTIONS

The following restrictions apply to this implementation of MS-Pascal:

1. Identifiers can have up to 31 characters. Longer identifiers are truncated.

2. Numeric constants can have up to 31 characters. Like identifiers, numeric constants longer than 31 characters are truncated.

3. LINK for this version of Pascal truncates global identifiers to 15 characters.

4. The PORT attribute for variables is identical to the ORIGIN attribute. It does not use I/O port addresses.

5. The maximum level to which procedures can be statically nested is 15. Dynamic nesting of procedures is limited by the size of the stack.

6. $SIMPLE currently turns off common subexpression optimization. $SIZE and $SPEED turn it back on (and have no other effect).

UNIMPLEMENTED FEATURES

The following MS-Pascal features are not presently implemented, or are implemented only as discussed below:

1. OTHERWISE is not accepted in RECORD declarations.

2. Code is generated for PURE functions, but no checking is done.

3. The extend level operators SHL, SHR, and ISR are not available.

4. The ENABIN, DISBIN, and VECTIN library routines are not available. The INTERRUPT attribute is ignored.

5. No checking is done for invalid GOTOs.

6. READ, READLN, and DECODE cannot have M and N parameters.

7. Enumerated I/O, permitting the reading and writing of enumerated constants as strings, is not available.

8. The metacommands $TAGCK, $STANDARD, $EXTEND, and $SYSTEM can be given, but have no effect.

9. The $INCONST metacommand does not accept string constants.

B. M20 PASCAL LIBRARY - FUNCTIONAL LIST

ABOUT THIS APPENDIX

This appendix lists the contents of the M20 PASCAL library in functional groups.

CONTENTS

BYTESTREAM I/O FUNCTIONS

Name	Parameter	Type
closedevice	did	WORD
closefile	did	WORD
directory	fileadr filelen	ADSMEM WORD
getbyte	did VAR retbyte	WORD BYTE
getlen	did VAR length	WORD INTEGER4
getposition	did VAR fposition	WORD INTEGER4
getstatus	did wordnum VAR parword	WORD WORD WORD
lookbyte	did VAR retbyte VAR bufstatus	WORD BYTE BYTE
opendevice	did	WORD
openfile	did mode extentlen filelen buffer	WORD WORD WORD WORD ADSMEM
peof	did VAR retstatus	WORD BOOLEAN
pseek	did fposition	WORD INTEGER4
putbyte	did inbyte	WORD BYTE
readbytes	did bytecount buffer VAR n_read	WORD WORD ADSMEM WORD

Name	Parameter	Type
readline	did	WORD
	bytecount	WORD
	buffer	ADSMEM
	VAR n_read	WORD
remove	fileadr	ADSMEM
	filelen	WORD
rename	oldfileadr	ADSMEM
	oldflen	WORD
	newfileadr	ADSMEM
	newflen	WORD
resetbyte	did	WORD
setcontrol	did	WORD
	wordnum	WORD
	parword	WORD
writebytes	did	WORD
	bytecount	WORD
	buffer	ADSMEM
	VAR n_write	WORD

BLOCK TRANSFER FUNCTIONS

Name	Parameter	Type
bclear	start	ADSMEM
	length	WORD
bmove	start	ADSMEM
	destination	ADSMEM
	length	WORD
bset	value	BYTE
	start	ADSMEM
	bytelen	WORD
bwset	wvalue	WORD
	start	ADSMEM
	wordlen	WORD

STORAGE ALLOCATION FUNCTIONS

Name	Parameter	Type
maxsize	VAR size	WORD
newabsanyseg	VAR tableadr size	ADSMEM WORD
newabsolute	VAR tableadr size	ADSMEM WORD
newlargestblock	VAR tableadr VAR size	ADSMEM WORD
newsameseg	VAR tableadr size	ADSMEM WORD
pdispose	VAR tableadr	ADSMEM
pnew	VAR tableadr size	ADSMEM WORD
stickynew	VAR tableadr size	ADSMEM WORD

TIME AND DATE FUNCTIONS

Name	Parameter	Type
getdate	dataadr length	ADSMEM WORD
gettime	dataadr length	ADSMEM WORD
setdate	dataadr length	ADSMEM WORD
settime	dataadr length	ADSMEM WORD

IEEE-488 FUNCTIONS

Name	Parameter	Type
ilninput	buffer	ADSMEM
	buflen	WORD
	talkeradr	WORD
	listenadr	WORD
	VAR bytes_not_read	WORD
ipoll	talkadr	WORD
	VAR statusptr	ADSMEM
iprint	buffer	ADSMEM
	listenadr	WORD
	buflen	WORD
	delimiter	WORD
iread	comlist	ADSMEM
	comlen	WORD
	buffer	ADSMEM
	buflen	WORD
ireset	(no parameters)	
iset	operand	WORD
isrq0	(no parameters)	
isrq1	(no parameters)	
iwrite	comlist	ADSMEM
	comlen	WORD
	numadr	ADSMEM
	numlen	WORD

ERROR PROCEDURE

Name	Parameter	Type
error	parnum	BYTE
	errorcode	BYTE

MISCELLANEOUS FUNCTIONS

Name	Parameter	Type
bootsys	(no parameters)	
checkvol	(no parameters)	
crlf	(no parameters)	
dhexbyte	byteval	WORD
dhexlong	longval	INTEGER4
dhexword	wordval	WORD
diskfree	volnum	WORD
	VAR secnum	INTEGER4
dlong	longval	INTEGER4
dnumw	wordval	INTEGER
	width	WORD
dstring	stringadr	ADSMEM
getvol	vbuffer	ADSMEM
	bufsize	WORD
	VAR vsize	WORD
parsename	stradr	ADSMEM
	strlen	WORD
	nameptr	ADSMEM
	VAR vol	WORD
sdevtab	devname	ADSMEM
	devlen	WORD
	VAR entrynum	BYTE
	VAR devtype	BYTE
	VAR tabptr	ADSMEM
search	drive	INTEGER
	mode	WORD
	infileptr	ADSMEM
	filelen	WORD
	outfileptr	ADSMEM
	VAR fileptr	ADSMEM
	VAR flen	WORD
	VAR blocknum	INTEGER4
setsysseg	(no parameters)	

Name	Parameter	Type
setvol	volnum	WORD
stringlen	stringptr VAR strlen	ADSMEM WORD

C. M20 PASCAL GRAPHICS LIBRARY - FUNCTIONAL LIST

ABOUT THIS APPENDIX

This appendix lists the contents of the M20 PASCAL graphics library in functional groups.

CONTENTS

TRANSFORMATION AND CONTROL

Functions:

Name	Parameter	Type
ClearViewArea	view_area_num	INTEGER
DivideViewArea	div_orient	INTEGER
	div_point	INTEGER
	VAR view_area_num	INTEGER
Escape	function_num	INTEGER
	ptr_datastruc	ADSMEM
SelectViewTrans	view_area_num	INTEGER
SetWorldCoordSp	view_area_num	INTEGER
	x0	REAL4
	y0	REAL4
	x1	REAL4
	y1	REAL4

Procedures:

Name	Parameter	Type
CloseGraphics	(no parameters)	
CloseViewTrans	view_area_num	INTEGER
OpenGraphics	(no parameters)	

GRAPHICS OUTPUT

Functions:

Name	Parameter	Type
GDP	func	INTEGER
	ptr_Xarray	ADSMEM
	ptr_Yarray	ADSMEM
	numb_of_points	INTEGER
	datarec	ADSMEM
GraphCursorAbs	x	REAL4
	y	REAL4
GraphCursorRel	dx	REAL4
	dy	REAL4
GraphPosAbs	x	REAL4
	y	REAL4
GraphPosRel	dx	REAL4
	dy	REAL4
LineAbs	x	REAL4
	y	REAL4
LineRel	dx	REAL4
	dy	REAL4
MarkerAbs	x	REAL4
	y	REAL4
MarkerRel	dx	REAL4
	dy	REAL4
PixelArray	x_width	REAL4
	y_height	REAL4
	ptr_array	ADSMEM

Name	Parameter	Type
Polyline	numb_of_points	INTEGER
	ptr_Xarray	ADSMEM
	ptr_Yarray	ADSMEM
Polymarker	num_points	INTEGER
	ptr_Xarray	ADSMEM
	ptr_Yarray	ADSMEM
TextCursor	column	INTEGER
	row	INTEGER

GRAPHICS ATTRIBUTES

Functions:

Name	Parameter	Type
SelectCursor	cursor_num	INTEGER
SelectGrColour	colour_code	INTEGER
SelectTxColour	fg_code	INTEGER
	bg_code	INTEGER
SetColourLogic	logop_code	INTEGER
SetColourRep	index	INTEGER
	colour	INTEGER
SetGrCsrBlnkrate	rate	INTEGER
SetGrCsrShape	ptr_array	ADSMEM
SetLineClass	class_num	INTEGER
SetTextline	chr_width	INTEGER
	textline_height	INTEGER
SetTxCsrBlnkrate	rate	INTEGER
SetTxCsrShape	ptr_array	ADSMEM

INQUIRY

Functions:

Name	Parameter	Type
ErrorInquiry	(no parameters)	
InqAttributes	VAR graphics_col	INTEGER
	VAR foregd_col	INTEGER
	VAR backgd_col	INTEGER
	VAR logic_oper	INTEGER
	VAR lineclass	INTEGER
InqCurTransNmbr	VAR view_area_num	INTEGER
InqGraphCursor	VAR x	REAL4
	VAR y	REAL4
	VAR blink_rate	INTEGER
InqGraphPos	VAR x	REAL4
	VAR y	REAL4
InqPixel	X_world_coord	REAL4
	Y_world_coord	REAL4
	VAR colour_num	INTEGER
InqPixelArray	X_width	REAL4
	Y_height	REAL4
	ptr_array	ADSMEM
	VAR invalid_code	INTEGER
InqPixelCoords	X_world_coord	REAL4
	Y_world_coord	REAL4
	VAR X_device_coord	INTEGER
	VAR Y_device_coord	INTEGER
InqTextCursor	VAR column	INTEGER
	VAR row	INTEGER
	VAR blink_rate	INTEGER
InqViewArea	VAR view_area_width	INTEGER
	VAR view_area_height	INTEGER
	VAR text_char_width	INTEGER
	VAR text_line_height	INTEGER
InqWorldCoordSp	VAR x0	REAL4
	VAR y0	REAL4
	VAR x1	REAL4
	VAR y1	REAL4

D. SYSTEM ERRORS

ERROR CODE (Decimal)	ERROR Description	ERROR CODE (Hexadecimal)
0	no error	00
2	syntax error	02
3	invalid termination of input bytestream	03
5	illegal function call	05
6	overflow	06
7	out of memory	07
9	EITHER invalid listener or talker address - when returned by an IEEE-488 function	09
	OR out of range - otherwise	
10	EITHER no IEEE board - when returned by an IEEE-488 function	0A
	OR duplicate definition - otherwise	
11	time out error	0B
13	type mismatch	0D
15	string too long	0F
18	undefined function	12
22	missing operand	16
23	buffer overflow	17
35	window not open	23

ERROR CODE (Decimal)	ERROR Description	ERROR CODE (Hexadecimal)
36	unable to create window	24
38	parameter out of range	26
53	file not found	35
54	bad file mode	36
55	file already open	37
57	disk i/o error	39
58	file already exists	3A
59	disk type mismatch	3B
60	disk not initialized	3C
61	disk filled	3D
62	end of file	3E
63	invalid record number	3F
64	invalid file name	40
67	too many files	43
68	internal error	44
69	volume name not found	45
70	rename error	46
71	invalid volume number	47

ERROR CODE (Decimal)	ERROR Description	ERROR CODE (Hexadecimal)
72	volume not enabled	48
73	invalid password	49
74	illegal disk change	4A
75	write protected file	4B
76	error in parameter	4C
77	invalid number of parameters	4D
78	file not open	4E
79	printer error	4F
80	copy protected file	50
81	paper empty	51
82	printer fault	52
92	command not found	5C
99	bad load file	63
101	error in time or date	65
108	call user error	6C
110	time out	6E
111	invalid device	6F

E. M20 - RS - 232 - C DEVICE PARAMETER TABLE

This appendix details the structure of the Device Parameter Table used by the two functions "getstatus" and "setcontrol". These functions are used for reading and writing device parameters for devices connected to the RS-232-C interfaces.

A knowledge of the hardware in question is useful for a deeper comprehension of this appendix (see M20 hardware literature).

WORD NUMBER	DESCRIPTION	
0-1	Ring buffer address	(long word)
2	Ring buffer input address	(word)
3	Ring buffer output address	(word)
4	Ring buffer count	(word)
5	Ring buffer size	(word)
6	75% of ring buffer size	(word)
7	50% of ring buffer size	(word)
8	8251A USART control port address	(word)
9	8251A USART state and error flags	(word)
10	8251A USART time out for data output	(word)
11 (high)	8251A USART mode	(byte)
11 (low)	8253 timer command	(byte)
12	8253 timer control port address	(word)
13	8253 timer baudrate data port address	(word)
14	8253 timer baud rate count	(word)
15	8259A PIC port A address	(word)
16	8259A PIC SEOI command word	(word)
17	8259A PIC - master interrupt mask bit	(word)
18	8259A U - slave interrupt mask bit	(word)

Word numbers 0 to 7 contain the state of the ring buffer. Words 8 to 11 (high) contain information relative to the 8251A (Programmable Communication Interface).

Word 8 contains the control port address. This can assume the following values:

%00C3 : USART motherboard control port.

%0803 : USART expansion board 1 control port.

%0823 : USART expansion board 2 control port.

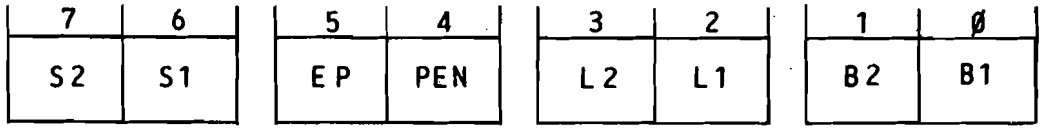
Word 9 represents the status and the error flags for the 8251 and is organised in the following way:

STATUS	BIT POSITION	LEGAL VALUES	MEANING
Duplex mode	15	1	full echoing of all input
		0	No echoing of input
(reserved)	14	0	(not used)
Framing Error	13	1	a valid stop bit has not been detected at the end of each character. (Reported from 8251A)
		0	No Framing Error
Overrun Error	12	1	a character has not been read before the next one becomes available. (Reported from 8251A)
		0	No Overrun Error
Parity Error	11	1	a change in parity value has been detected. (Reported from 8251A)
		0	No Parity Error
Timeout Error	10	1	a timeout has occured while waiting for the Transmit Ready line on the 8251A
		0	No Timeout Error
Memory Error	9	1	driver failed to open to open buffer - no Open Port call or insufficient memory.
		0	No Memory Error
Buffer Error	8	1	interrupt routine tried to overwrite the buffer.
		0	No Buffer Error

STATUS	BIT POSITION	LEGAL VALUES	MEANING
(reserved)	7	0	(not used)
Free-running protocol	6	1 0	free-running protocol, Handshake protocol using XON/XOFF
XOFF/XON Flag (M20 previously acted as trans-mitter)	5	1 0	XOFF character, sent in previous trans-mission. Buffer is 75% full. XOFF is sent from M20 i.e. other sender should stop. XON character, sent in previous trans-mission. Input buffer is ready to receive characters (default state.) XON is sent from M20 i.e. other sender should start again.
Hardware State	4	1 0	hardware present and 8259A passed interrupt mask test. No hardware or failed test
XOFF/XON	3	1 0	XOFF character, de-tected in current reception. XOFF character is received from outside. No characters will be transmitted. XON character, de-tected in current reception. XON received from outside. Characters will be transmitted (default state).
(reserved)	2	0	(not used)
(reserved)	1	0	(not used)
(reserved)	0	0	(not used)

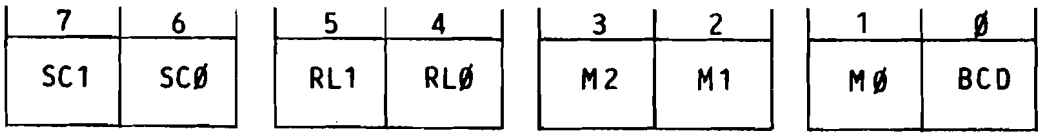
Word 10 contains the time-out value for the transmission of data.

The high byte in word 11 is the 8251A Mode byte and is described below:



Number of Stop Bits:	S2	S1		
	0	1	1 stop bit	
	1	0	1.5 stop bits	
	1	1	2 stop bits	(default)
	0	0	ILLEGAL	
Even Parity/ Parity Enable:	EP	PEN		
	0	0	Disable Parity/Odd Parity	(default)
	0	1	Enable Parity/Odd Parity	
	1	1	Enable Parity/Even Parity	
	1	0	Disable Parity/Even Parity	
Character Length:	L2	L1		
	0	0	5 Data bits	
	0	1	6 Data bits	
	1	0	7 Data bits	(default)
	1	1	8 Data bits	
Baud Rate Factor:	B2	B1		
	1	0	Asynchronous Mode 16 x	(default)
	0	0	Synchronous Mode	
	0	1	Asynchronous Mode 1 x	
	1	1	Asynchronous Mode 64 x	!

The low byte in word 11, and words 12 to 14 concern the 8253 timer (Programmable interval timer). The low byte in word 11 is the 8253 command byte described below:



Counter Select:		SC1	SC0	
		0	0	Select Counter 0
		0	1	Select Counter 1
		1	0	Select Counter 2
		1	1	ILLEGAL
Read/Load Instruction:		RL1	RL0	
		0	0	Counter Latching Operation
		0	1	Read/Load most sig. byte only (msb)
		1	0	Read/Load least sig. byte only (lsb)
		1	1	Read/Load lsb first, then msb
Mode:	M2	M1	M0	
	0	0	0	Mode 0: Interrupt on Terminal Count
	0	0	1	Mode 1: Programmable One-Shot
	x	1	0	Mode 2: Rate Generator
	x	1	1	Mode 3: Square Wave Rate Generator
	1	0	0	Mode 4: Software Triggered Strobe
	1	0	1	Mode 5: Hardware Triggered Strobe
4 BCD's/ Binary Word		BCD		
		0		Binary Counter (16 bits)
		1		BCD Counter (4 decades * 4 bits/ decade)

Word 12 contains the 8253 control port address; this can be either

%0127	motherboard timer control port
%0867	expansion board timer control port

Word 13 contains a channel address of an 8253 timer. The address can be one of the following:

%0121	channel 0 motherboard timer
%0123	channel 1 motherboard timer
%0125	channel 2 motherboard timer
%0861	channel 0 expansion board timer
%0863	channel 1 expansion board timer
%0865	channel 2 expansion board timer

Word 14 sets the transmission baud rate as follows:

1538	baud count for baud rate of 50
699	baud count for baud rate of 110
256	baud count for baud rate of 300
128	baud count for baud rate of 600
64	baud count for baud rate of 1200
32	baud count for baud rate of 2400
16	baud count for baud rate of 4800
8	baud count for baud rate of 9600
4	baud count for baud rate of 19200

Word 15 contains the 8259 control port address (Prgrammable Interrupt Controller (PIC)). These can be:

%0140	mother board PIC control port address
%0840	expansion board PIC control port A address

Note that even addresses for programmable interrupt controller B data port addresses are assumed to be 2 more than A control port addresses.

Word 16 contains the SEOI (Specific End Of Interrupt) command to be issued before exiting the interrupt routine. The SEOI is calculated using the formula:

SEOI = %C0 + (2* IR No.)

where IR No.is an interrupt routine number from 0 - 7.

The RS-232-C SEOI's are the following:

%00C6	master 8259A pic SEOI for IR3 (tty mother)
%00CE	master 8259A pic SEOI for IR7 (expansion)
%00C0	slave 8259A pic SEOI for IR0 (port 1)
%00C4	slave 8259A pic SEOI for IR2 (port 2)

The following table gives all the M20 interrupt assignments.

Master 8259A PIC Mother Board Interrupt Assignments:

```
IR0:     Floppy Disk Controller
IR1:     External Daisy Chain Request   (potentially a slave 8259A)
IR2:     External Daisy Chain Request   (potentially a slave 8259A)
IR3:     RxD: DTE TTY/Remote    8251A
IR4:     RxD: keyboard          8251A
IR5:     TxD: DTE/TTY/Remote    8251A   (not used)
IR6:     Parallel 8255A PC0 or PC3
IR7:     External Daisy Chain Request   (used w/ RS-232-C Expansion Board)
```

Slave 8259A PIC Expansion Board Interrupt Assignments:

```
IR0:     RxD: DTE/TTY port 1/Remote      8251A
IR1:     TxD: DTE/TTY port 1/Remote      8251A      (not used)
IR2:     RxD  DTE/TTY port 2/Remote      8251A
IR3:     TxD: DTE/TTY port 2/Remote      8251A      (not used)
IR4:     grounded (not used)
IR5:     grounded (not used)
IR6:     grounded (not used)
```

Words 17 and 18 contain the masks relative to the interrupt levels. The mask values are the following:

8259A PIC Interrrupt Assignments (by bit with data bus shift):

```
%0100     IR7 interrupt mask
%0080     IR6 interrupt mask
%0040     IR5 interrupt mask
%0020     IR4 interrupt mask
%0010     IR3 interrupt mask
%0008     IR2 interrupt mask
%0004     IR1 interrupt mask
%0002     IR0 interrupt mask
```

F. DEVICE ID (DID) ASSIGNMENTS

DEVICE ID (DID) ASSIGNMENTS

The following table lists all the DID assignments. Some of these DID's represent devices which are always open, others are assigned by functions of the M20 PASCAL library.

1	BASIC files
2	.
.	.
.	.
15	.
17	Console
18	Printer
19	Communications RS-232-C
20	System Disk Files (not accessible to BASIC)
.	.
.	.
.	.
24	.
25	Com1 (RS-232-C)
26	Com2 (RS-232-C)

G. VOCABULARY

This appendix reviews some of the vocabulary that is commonly used in discussing the steps in program development. The definitions given are intended primarily for use with this user guide. Thus, neither the individual definition nor the list of terms is comprehensive.

Some other terms you should know are related to stages in the development and execution of a compiled program. These stages are:

1. Compile time

   The time during which the compiler is executing and during which it compiles a source file and creates a relocatable object file.

2. Link time

   The time during which the linker is executing and during which it links together relocatable object files and library files.

3. Runtime

   The time during which a compiled and linked program is executing. By convention, runtime refers to the execution time of your program and not to the execution time of the compiler or the linker.

The following terms pertain to the linking process and the runtime library:

1. Module

   A general term for a discrete unit of code. There are several types of modules, including relocatable and executable modules. (Furthermore, in the Olivetti PASCAL language, "module" has a specific meaning as one type of Olivetti PASCAL compiland. See the "PASCAL Reference Manual" for details. In this User Guide, we use the term "module" in its general sense, unless otherwise specified.)

   The object files created by the compiler are said to be "relocatable," that is, they do not contain absolute addresses. Linking produces an "executable" module, that is, one that contains the necessary addresses to proceed with loading and running the program.

2. Routine

   Code, residing in a module, that represents a particular procedure or function. More than one routine may reside in a module.

3. External reference

   A variable or routine in one module that is referred to by a routine in another module.

   The variable or routine is often said to be "defined" or "public" in the module in which it resides.

   The linker tries to resolve external references by searching for the

declaration of each such reference in other modules. If such a declaration is found, the module in which it resides is selected to be part of the executable module (if it is not already selected) and becomes part of your executable file. These other modules are usually library modules in the runtime library.

If the variable or routine is found, the address associated with it is substituted for the reference in the first module, which is then said to be "bound." When a variable is not found, it is said to be "undefined" or "unresolved."

4. Relocatable module

The module's code can be loaded and run at different locations in memory. Relocatable modules contain routines and variables represented as offsets relative to the start of the module. These routines and variables are said to be at "relative" offset addresses. When the module is processed by the linker, an address is associated with the start of the module.

The linker then computes an absolute offset address that is equal to the associated address plus the relative offset for each routine or variable. These new computed values become the absolute offset addresses that are used in the executable file. Compiled object files and library files are all relocatable modules.

These offset addresses are still relative to a "segment," which corresponds to an Z8000 segment register. Segment addresses are not defined by the linker; rather, they are computed when your program is actually loaded prior to execution.

5. Runtime library

Contains the runtime routines needed to implement the M20 Pascal language. A library module usually corresponds to a feature or subfeature of the M20 Pascal language.

H. ASCII CODE

This table shows decimal, hexadecimal, and binary representation of the ASCII code. (Boxed characters are different on national keyboards.)

a	b	c	d	a	b	c	d	a	b	c	a	b	c
0	00	0000 0000	NUL	64	40	0100 0000	@	128	80	1000 0000	192	C0	1100 0000
1	01	0000 0001	SOH	65	41	0100 0001	A	129	81	1000 0001	193	C1	1100 0001
2	02	0000 0010	STX	66	42	0100 0010	B	130	82	1000 0010	194	C2	1100 0010
3	03	0000 0011	ETX	67	43	0100 0011	C	131	83	1000 0011	195	C3	1100 0011
4	04	0000 0100	EQT	68	44	0100 0100	D	132	84	1000 0100	196	C4	1100 0100
5	05	0000 0101	ENQ	69	45	0100 0101	E	133	85	1000 0101	197	C5	1100 0101
6	06	0000 0110	ACK	70	46	0100 0110	F	134	86	1000 0110	198	C6	1100 0110
7	07	0000 0111	BEL	71	47	0100 0111	G	135	87	1000 0111	199	C7	1100 0111
8	08	0000 1000	BS	72	48	0100 1000	H	136	88	1000 1000	200	C8	1100 1000
9	09	0000 1001	HT	73	49	0100 1001	I	137	89	1000 1001	201	C9	1100 1001
10	0A	0000 1010	LF	74	4A	0100 1010	J	138	8A	1000 1010	202	CA	1100 1010
11	0B	0000 1011	VT	75	4B	0100 1011	K	139	8B	1000 1011	203	CB	1100 1011
12	0C	0000 1100	FF	76	4C	0100 1100	L	140	8C	1000 1100	204	CC	1100 1100
13	0D	0000 1101	CR	77	4D	0100 1101	M	141	8D	1000 1101	205	CD	1100 1101
14	0E	0000 1110	SO	78	4E	0100 1110	N	142	8E	1000 1110	206	CE	1100 1110
15	0F	0000 1111	SI	79	4F	0100 1111	O	143	8F	1000 1111	207	CF	1100 1111
16	10	0001 0000	DLE	80	50	0101 0000	P	144	90	1001 0000	208	D0	1101 0000
17	11	0001 0001	DC1	81	51	0101 0001	Q	145	91	1001 0001	209	D1	1101 0001
18	12	0001 0010	DC2	82	52	0101 0010	R	146	92	1001 0010	210	D2	1101 0010
19	13	0001 0011	DC3	83	53	0101 0011	S	147	93	1001 0011	211	D3	1101 0011
20	14	0001 0100	DC4	84	54	0101 0100	T	148	94	1001 0100	212	D4	1101 0100
21	15	0001 0101	NAK	85	55	0101 0101	U	149	95	1001 0101	213	D5	1101 0101
22	16	0001 0110	SYN	86	56	0101 0110	V	150	96	1001 0110	214	D6	1101 0110
23	17	0001 0111	ETB	87	57	0101 0111	W	151	97	1001 0111	215	D7	1101 0111
24	18	0001 1000	CAN	88	58	0101 1000	X	152	98	1001 1000	216	D8	1101 1000
25	19	0001 1001	EM	89	59	0101 1001	Y	153	99	1001 1001	217	D9	1101 1001
26	1A	0001 1010	SUB	90	5A	0101 1010	Z	154	9A	1001 1010	218	DA	1101 1010
27	1B	0001 1011	ESC	91	5B	0101 1011	[	155	9B	1001 1011	219	DB	1101 1011
28	1C	0001 1100	FS	92	5C	0101 1100	\	156	9C	1001 1100	220	DC	1101 1100
29	1D	0001 1101	GS	93	5D	0101 1101	]	157	9D	1001 1101	221	DD	1101 1101
30	1E	0001 1110	RS	94	5E	0101 1110	↑	158	9E	1001 1110	222	DE	1101 1110
31	1F	0001 1111	US	95	5F	0101 1111	—	159	9F	1001 1111	223	DF	1101 1111
32	20	0010 0000	SPACE	96	60	0110 0000	`	160	A0	1010 0000	224	E0	1110 0000
33	21	0010 0001	!	97	61	0110 0001	a	161	A1	1010 0001	225	E1	1110 0001
34	22	0010 0010	"	98	62	0110 0010	b	162	A2	1010 0010	226	E2	1110 0010
35	23	0010 0011	#	99	63	0110 0011	c	163	A3	1010 0011	227	E3	1110 0011
36	24	0010 0100	$	100	64	0110 0100	d	164	A4	1010 0100	228	E4	1110 0100
37	25	0010 0101	%	101	65	0110 0101	e	165	A5	1010 0101	229	E5	1110 0101
38	26	0010 0110	&	102	66	0110 0110	f	166	A6	1010 0110	230	E6	1110 0110
39	27	0010 0111	'	103	67	0110 0111	g	167	A7	1010 0111	231	E7	1110 0111
40	28	0010 1000	(	104	68	0110 1000	h	168	A8	1010 1000	232	E8	1110 1000
41	29	0010 1001	)	105	69	0110 1001	i	169	A9	1010 1001	233	E9	1110 1001
42	2A	0010 1010	*	106	6A	0110 1010	j	170	AA	1010 1010	234	EA	1110 1010
43	2B	0010 1011	+	107	6B	0110 1011	k	171	AB	1010 1011	235	EB	1110 1011
44	2C	0010 1100	,	108	6C	0110 1100	l	172	AC	1010 1100	236	EC	1110 1100
45	2D	0010 1101	-	109	6D	0110 1101	m	173	AD	1010 1101	237	ED	1110 1101
46	2E	0010 1110	.	110	6E	0110 1110	n	174	AE	1010 1110	238	EE	1110 1110
47	2F	0010 1111	/	111	6F	0110 1111	o	175	AF	1010 1111	239	EF	1110 1111
48	30	0011 0000	0	112	70	0111 0000	p	176	B0	1011 0000	240	F0	1111 0000
49	31	0011 0001	1	113	71	0111 0001	q	177	B1	1011 0001	241	F1	1111 0001
50	32	0011 0010	2	114	72	0111 0010	r	178	B2	1011 0010	242	F2	1111 0010
51	33	0011 0011	3	115	73	0111 0011	s	179	B3	1011 0011	243	F3	1111 0011
52	34	0011 0100	4	116	74	0111 0100	t	180	B4	1011 0100	244	F4	1111 0100
53	35	0011 0101	5	117	75	0111 0101	u	181	B5	1011 0101	245	F5	1111 0101
54	36	0011 0110	6	118	76	0111 0110	v	182	B6	1011 0110	246	F6	1111 0110
55	37	0011 0111	7	119	77	0111 0111	w	183	B7	1011 0111	247	F7	1111 0111
56	38	0011 1000	8	120	78	0111 1000	x	184	B8	1011 1000	248	F8	1111 1000
57	39	0011 1001	9	121	79	0111 1001	y	185	B9	1011 1001	249	F9	1111 1001
58	3A	0011 1010	:	122	7A	0111 1010	z	186	BA	1011 1010	250	FA	1111 1010
59	3B	0011 1011	;	123	7B	0111 1011	{	187	BB	1011 1011	251	FB	1111 1011
60	3C	0011 1100	<	124	7C	0111 1100	|	188	BC	1011 1100	252	FC	1111 1100
61	3D	0011 1101	=	125	7D	0111 1101	}	189	BD	1011 1101	253	FD	1111 1101
62	3E	0011 1110	>	126	7E	0111 1110	~	190	BE	1011 1110	254	FE	1111 1110
63	3F	0011 1111	?	127	7F	0111 1111	DEL	191	BF	1011 1111	255	FF	1111 1111