THE BASIC COMPILER

PETSPEED 128

UP TO 40 TIMES THE SPEED OF BASIC

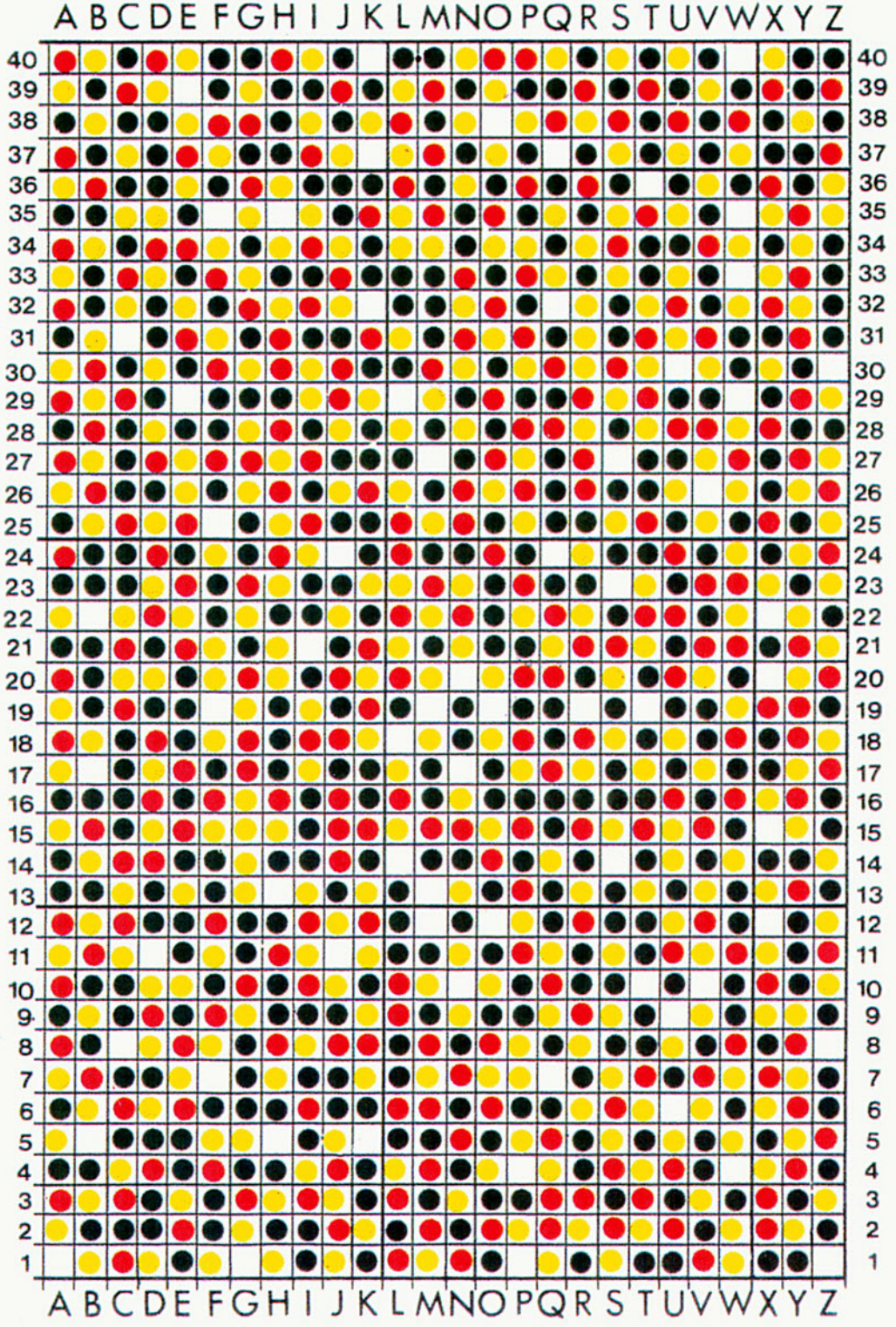
ALL BASIC 7 COMMANDS PLUS

MAKES LONG PROGRAMS SHORTER

EXTENSIONS TO BASIC

EASY TO USE

COMMODORE 128


A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

COPYRIGHT

Petspeed 128 is copyrighted and all rights to it are reserved by SYSTEM SOFTWARE OXFORD LTD. The product is intended for use by the original purchaser only. The purchaser of Petspeed 128 is hereby licensed only to read the software from its medium into the memory of the Commodore 128 computer solely for the purpose of executing it therein. Duplicating, copying (other than for backup purposes), selling or otherwise distributing this product is a violation of the law.
It is the policy of the manufacturers to vigorously pursue litigation against all infringements of their copyright.
Having said this we hope you will enjoy using Petspeed and that you will find it the useful utility that it is intended to be.

DISCLAIMER

Although Petspeed 128 has been thoroughly tested, no claim is made by the authors or manufacturers concerning the adherance of the software to any particular specification or the suitability of Petspeed 128 for any particular purpose.

CONTENTS

Hardware

In order to run Petspeed 128 you will need

1 A Commodore 128 computer

2 A 1571 or 1541 disk drive

Product Description

The Petspeed 128 product comprises the following

1 This manual

2 A 5.25 inch floppy diskette (known as the system disk) containing the compiler and associated files.

3 A protection sheet showing a matrix of coloured squares. This sheet is either enclosed in the manual or printed on the front inside cover of the manual.

Diskette Care

The method used to protect Petspeed (sorry but we must do this), unlike other protection methods, does at least allow you to make backup copies of the system disk. You are advised to make backups and to keep them in a safe place. All the usual rules apply to the care of the system disk; don't sit on it, don't eat lunch on it and so on.

Introduction

Petspeed was first released in the early days of Commodore computing and has been implemented on most of the major Commodore machines. Petspeed offers a very high level of compatibility with BASIC 7.0. It has always been and continues to be the BASIC compiler which produces the fastest running compiled programs.

Petspeed 128 is a very significant enhancement of earlier versions in that it allows full use of all the 128's memory and supports the whole of the 128's very powerful BASIC language.

A section of the manual devoted to the technical details of Petspeed compiled programs provides all information needed to interface machine code routines. A special facility allows the compiler to handle extensions to BASIC of the sort provided by certain commercial software packages.

Readers of this manual are expected to be familiar with BASIC 7.0 and its resident interpreter. No attempt is made to teach BASIC programming to the novice. The manual only discusses aspects of BASIC which are affected by compilation. A glossary of terms is provided.

A Word Of caution

Petspeed is so simple to use that there is a temptation for programmers to skim through or even to completely ignore the manual. This is a practice which we strongly discourage. Users are urged to spend a little time studying the manual thoroughly. Sections of particular importance are **Compatibility with BASIC 7.0, Disks and Formats** and **Errors.**

Compiling A Program

Compiling programs under Petspeed has been made as straightforward as possible. Petspeed is loaded into the 128 from the system disk and then used to process an ordinary BASIC program file to produce a loadable object module on the same disk. When the compilation has finished the compiled program is left, ready to run, in the computer's memory.

Before using Petspeed it is necessary to follow a brief protection sequence to establish you as an authorised user of the software. We hope that you will not find the protection too tiresome; software does have to be protected and this method does at least allow you to make backup copies of the system disk.

To compile a program under Petspeed 128 you will need two disks

1 The Petspeed System Disk supplied with this manual.

2 A work disk containing the source program.

During the compilation process, Petspeed writes several large temporary files to the work disk which should contain enough free space to accommodate them. For best results the work disk should be empty apart from the source program.

Once the work disk has been prepared you are ready to compile. Proceed as follows:

1 Mount the Petspeed System disk and type SHIFT RUN/STOP.
At this point Petspeed makes the protection check. On the inside cover of this manual or on a separate sheet will be found a matrix of colored squares. Each square can be identified by means of a simple grid reference. For example to find square E-7, identify column E (marked along the bottom of the matrix) and row 7 (marked along the left hand edge). Square E-7 is where column E and row 7 meet. (see fig 1).

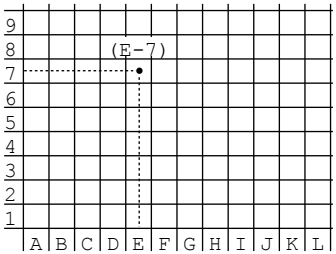


Fig 1.

The protection check is very simple; all you have to do is correctly identify three squares and enter their colors. Petspeed will give instructions as follows:

WHAT COLOR IS SQUARE X-XX ?

where X-XX is a grid reference. When you have found the square, enter one of the letters W,Y,G or R depending upon whether the square is white, yellow, green or red. When you have correctly answered three such questions the protection check is complete.

2 When requested, type in the name of your source program which should not be more than eleven characters long. At this point Petspeed loads a number of files from the System disk. After a short time the message

MOUNT WORK DISK ... HIT ANY KEY

will appear. Obey.

Now sit back and wait. Petspeed is a four pass compiler which means that it reads the the source program (or some representation of it) four times. The four passes are necessary because of the amount of optimisation carried out by Petspeed and the compilation can take a little time to run. Most of the time the compiler lets you know what it's doing by printing the line numbers of the source program as it is being processed.

WARNING If the compiler does not behave as de scribed and in particular, stops during pass3 with a PARSE TREE ERROR, you have probably encountered a problem with the 1571 disk drive. Read the section on **Disks and formats**.

When Petspeed is through, the screen will display certain information about the program that has just been compiled. This information includes a count of the number of lines, variables and arrays and the sizes of the various segments. For more information see the section on interfacing machine code to Petspeed.

At the bottom of the screen the cursor will re-appear over the a DSAVE command. If you wish to save the compiled program to disk, simply hit <return>, to run type RUN.

Petspeed ends with the compiled program in memory. If you choose to save the program it will have the same name as the source program with the characters .BRON appended, you can give it a different name by editing the BSAVE command. A compiled program can be loaded and run just like a regular BASIC program. If you LIST however all you will see is

```
10 BANK(15):SYS(40783)
20 BANK(00):SYS(16452):PETSPEED
```

This is entirely in order. The SYS(40783) ensures that the bit map screen is allocated and the address 16452 is the entry point for the Petspeed RTS which, once called retains control while the program is running.

Running compiled programs

Compiled programs are executed just like BASIC programs simply by typing RUN. They may themselves load compiled or BASIC programs and BASIC programs may load compiled programs without restriction.

Copying compiled programs

Any method which can be used to save a BASIC program can also be used to save a compiled program. The simplest method is to load the compiled program that you wish to save into memory using DLOAD and save it again using DSAVE.

Fast and slow mode

Petspeed itself (and of course compiled programs) can make use of either the 40 or the 80 column screen in either FAST (2MHz) or SLOW (1MHz) mode. Unfortunately, whenever FAST mode is selected, the 40 column screen collapses and because this makes it difficult to check the progress of a compilation, Petspeed normally runs in FAST mode with an 80 column screen and in SLOW mode with a 40 column screen.

The fast and slow directives

If you wish to change these default settings and have Petspeed use FAST mode with a 40 column screen or SLOW mode with an 80 column screen you can do so as follows. To force FAST mode, follow the filename (when Petspeed first requests it) with a ,F. To force SLOW mode follow the filename with a ,S. It is well worthwhile to compile in FAST mode on a 40 column screen even though you cannot see what is going on. Petspeed will automatically go into SLOW mode and re-enable the screen at the end of compilation or in the event of an error.

Compatibility With BASIC 7.0

Petspeed 128 aims for maximum compatibility with BASIC 7.0. There are some trivial differences but these can be allowed for very easily. They fall into two groups: Facilities not available under Petspeed and extra facilities provided by Petspeed.

Facilities not available under Petspeed

1 When a program is overlayed all variables are normally cleared. For information on overlaying with variables preserved, see the section on overlaying.

2 Petspeed needs to know at compile time the dimensions of all arrays. If an array is not dimensioned explicitly in a program it will, as it is in BASIC, be dimensioned to eleven elements times the number of subscripts. If an array is dimensioned however it must be dimensioned with constants. Although this slight restriction may sometimes be a problem, it does give Petspeeded programs the ability to access array elements much more rapidly than would otherwise be possible. Most programs spend a high proportion of their run time accessing array elements and it is a great advantage for a compiler to be able to allocate array space and fix addresses during compilation. For example a statement like

```
100 DIM A(N)
```

is illegal in Petspeed because the compiler cannot know had much store to reserve. Instead something like the following should be substituted.

```
100 DIM A(50)
```

If this is not done, the compiler will stop during pass l and ask for the dimension as follows:

```
LINE 100: DIM A(
```

Once the dimension is typed in, compilation will continue.

3 Certain BASIC statements such as RESTORE and COLLISION which take a line number as argument allow an expression instead of a constant line number. Petspeed does not allow this. A statement like RESTORE N+1 for example should therefore be replaced with something like RESTORE 100.

4 The EL reserved variable, after a TRAP contains not the line number where the error occured but the equivalent Speedcode address. For more information see the section on ERRORS.

5 Petspeed does not support RESUME NEXT. RESUME <line-number> IS allowed.

Extra programming facilities

1 User defined string and mixed functions are allowed. Either the function itself or its argument or both may be of either string or numeric type. That is, all the following are possible

```
DEF FN A(X),   DEF FN A$(X),   DEF FN A(X$),   DEF FN A$(X$)
```

2 Integer FOR loops such as the following are allowed:

```
FOR I%=0 TO 10: ..... NEXT I%
```

There is a slight speed advantage in using integer FOR loops. This is because Petspeed can be sure at compile time that the loop variable will not have a real value assigned to it. The speed advantage is only slight because Petspeed will try to run all FOR loops as integer loops anyway, only converting to floating point if it is forced to do so.

3 In BASIC 7.0, variable, array and function names can be of any length. The interpreter does however only take notice of the first two characters together with any type descriptor (% or $). To maintain compatibility, Petspeed treats names in the same way but provides the option of making all characters significant. The following compiler directive is provided for this purpose.

```
REM !LN
```

This must appear at the beginning of a program before any variables, arrays or functions are defined.

4 Under Petspeed, the STOP key is disabled by default. Two statements are provided to enable and disable the STOP key when required; these are as follows:

```
REM !ES       (enables the STOP key)
REM !DS       (disables the STOP key)
```

These are statements and NOT compiler directives, that is they will be executed at run time. In the following

```
10 REM !ES
20 GOSUB 100
30 REM !DS
```

the subroutine at line 100 will be executed with the stop key enabled.

The REM !DS statement is to be preferred over other methods of disabling the STOP key in that it has no side effects. Programs also run slightly faster with STOP disabled.

5 The & command.

BASIC 7.0 provides the command CMD in order to re-direct the 128's standard output to a file. The use of CMD can often speed up file writing as in the following code fragment which writes 101 strings to disk.

```
10 OPEN 5,8,8,"0:diskfile,s,w"
20 CMD 5
30 FOR I=0 to 100
40 PRINT "testdata"
50 NEXT
60 PRINT#5
70 CLOSE 5
```

Unfortunately BASIC 7.0 provides no equivalent command to re-direct the 128's standard input. This is the purpose of the & command. The statement

```
&F
```

where F is the logical file number of an open read file, will cause INPUT or GET to accept input from the file associated with F until the next INPUT#F or GET#F statement is executed. For example, here is a program to read back the data written in the example above:

```
10 OPEN 5,8,8,"0:diskfile,s,r"
20 &5
30 FOR I=0 to 100
40 INPUT A$:PRINT A$
50 NEXT
60 GET#5,a$
70 CLOSE 5
```

6 The % directive

This directive is provided to allow Petspeed to handle extensions to BASIC provided by commercial packages or user written machine code.

Extended statements must take the form of a keyword optionally followed by a number of parameters. For example, suppose that an extended BASIC package provides the command

```
CLEAR S,N
```

where S and N are expressions specifying the start and size of an area of memory to be cleared to zero. In the normal way, Petspeed would throw this out as a syntax error. By preceding the statement with a % sign however, the run time system can be instructed to have that statement alone executed under the interpreter. for example

```
%CLEAR b+l,a(i)   would be perfectly acceptable
```

Even though the Petspeed run time system does call the interpreter to execute such commands, because Petspeed variables are held in a special format, the arguments of extended commands are subjected to some processing at compile time and only certain command formats are allowed.

In general commands proceeded by % must consist of an optionally tokenised keyword followed by up to ten parameters separated by commas, semicolons or by the BASIC words TO and ON.

Making the most of Petspeed 128

Petspeed is an optimising compiler. In general, it is the job of Petspeed to make the most of your programs rather than the other way around. For example, although Petspeed makes extensive use of integer arithmetic, there is no command to convert variables to integer type explicitly as the compiler does this automatically.

The way to make the most of Petspeed is to exploit the freedom it gives to dispense with all the awkward programming practices that are ordinarily necessary to maximise the speed of the interpreter.

There is no longer any need to assign frequently occuring variables at the beginning of a program. Petspeed does treat these variables in a special way but it can find them for itself without the help of the programmer.

There is no need to locate frequently called subroutines near the beginning of a program. Petspeed calls all subroutines (and GOTO targets) directly at maximum speed.

There is no need to pack many statements on to the same line. Object programs contain no line numbers.

There is no need to use variables rather than constants (which practice normally speeds up the interpreter). All constants are converted to binary at compile time.

On the whole it is very hard to slow Petspeed down by using techniques that make programs readable. When programming with Petspeed in mind, forget all about execution efficiency and concentrate on producing good maintainable code.

Errors

Although Petspeed checks programs at compile time and at run time for most types of error, it is strongly recommended that program development is done under the BASIC interpreter where errors can be corrected interactively. It is to be hoped that by the time programs are submitted to Petspeed they will be free from elementry syntax errors.

Two types of error are possible: errors that occur during compilation (compile time errors) and errors that occur when a program is run (run time errors).

Compile time errors

Most the usual BASIC 7.0 errors may occur during compilation and most of these will cause compilation to abort with a suitable message. The only type of compile time error which will not cause the compiler to abort corresponds to BASIC 7.0's UNDEF'D STATEMENT ERROR. Such errors will be found during pass 4 when the line numbers concerned will appear in reverse field. A program containing references to undefined statements will run, provided that no statement containing such a reference is ever executed. If such a statement is executed the run will terminate with message:

UNDEF'D STATEMENT ERROR IN XXXXX

where XXXXX is a LINE NUMBER.

An error that may occur at compile time where it would not have done in BASIC 7.0 is FORMULA TOO COMPLEX ERROR. This may happen if the 3rd argument of a MID$ or INSTR contains a function or array references. The problem may be easily solved by replacing the 3rd argument with a variable.

Note that occasionally the line number given in a compile time error report is off by one line, that is, the error actually occurred in the line preceding the one reported.

When compilation is aborted due to a compile time error, certain work files are left on the directory. These may be scratched or left on the disk where they will be overwritten when the program is re-compiled.

Run Time Errors

With a single exception, compiled programs will report all errors that occur at run time. The exception is BAD SUBSCRIPT ERROR. Because of the considerable checking overhead that would be involved, the Petspeed RTS does not check that subscripts are in range. If a compiled program does try to access array elements outside the declared (or default) dimensions, the results will be unpredictable.

The ERRORS utility program

When an error occurs at run time or when a STOP statement is executed, the Petspeed RTS will print a message identical to the one that would have appeared had the program been running under BASIC. The UNDEF'D STATEMENT ERROR and BREAK messages are foliaged by line numbers just as they are under BASIC. Other messages refer not to line numbers but to speedcode addresses which have little meaning for the programmer. A special utility ERRORS is provided on the system disk to convert speedcode addresses into line numbers. The utility makes use of a file generated by Petspeed, which contains a list of source program lines together with the corresponding Speedcode addresses. This file has the same name as the source program but with the characters .ER appended. We illustrate the use of the ERRORS utility by example.

Suppose that you have successfully compiled a program AH to produce the object program AH.BRON. When the program is run it produces the message.

?OUT OF DATA IN 40010

The number 40010 is not a line number but a speedcode address. To find the line in which the error actually occurred proceed as follows.

1 Make a note of the speedcode address (in this case 40010).

2 Mount the system disk and RUN "ERRORS".

3 When requested, enter the name of the program that caused the error (in this case AH) and the speedcode address as given in the error report. ERRORS will now request you to mount the work disk containing the .ER file. After a few seconds something like the following will appear.

100 xxxxxxxXxxxxxxxxxxxxxxxxxxxxxxxxx

That is, the line number of the line containing the code which caused the error followed by a string of x's. The x's actually represent the speedcode equivalent of that line of BASIC with the position at which the error occurred highlighted. The correspondence will not be exact but the report will give you a good idea of where to look in a long line of code.

Compiler errors

There is a third class of error messages which allude to built in limits such as the maximum number of lines, variables, arrays etc which are acceptable by the compiler. These limits are interdependent and vary from version to version of Petspeed. They are however set to high values and should not normally be exceeded by any program which will run in BASIC. If this type of error should occur, the message produced will give sufficient information to allow you to solve the problem.

The EL reserved variable

The object program generated by Petspeed has no knowledge of the the source program's line numbers. In the same way that run time errors generate a speedcode address (see above) EL, after a TRAP will contain the speedcode address where the error occured. The corresponding line number may be obtained by use of the ERRORS utility.

Overlaying

BASIC 7.0 allows programs to be overlayed with the data from the previous program still intact. Petspeed in default mode does not behave like this. In the absence of a command to the contrary, compiled programs clear all variables upon entry. Petspeed can compile programs which will overlay with data intact but this facility is restricted as follows.

1 Only arrays may be passed to an overlaid program.

2 The overlaid program must contain exactly the same array declarations as the overlaying program in the same order.

3 The overlaid program must contain the compiler directive REM !ol.

Suppose that we have two programs A and B. A runs for a while and then loads B. We wish B to make use of data placed in arrays by A. We proceed as follows:

1 First we must make sure that A and B contain the same array declarations. This means that all arrays should be DIMmed and that exactly the same DIM statements should appear in A and B as the first statements in the program. B may contain additional arrays not declared in A but these must be DIMmed after the arrays known to A.

2 Next we must include the compiler directive REM !ol

somewhere in B. This tells Petspeed to generate an object program which does not clear arrays on entry.

3 A and B may now be compiled.

In many cases, a program (for example a menu program) will overlay any one of a number of other programs. In such a case all the programs which could be overlaid should be treated in exactly the same way as B.

When you are compiling programs which call overlays and on other occasions, you may find it useful to be able to test inside a program whether or not the program is compiled. This is simple. if PEEK(762)=96 then the program is compiled, otherwise it is not.

Disks And Formats

Petspeed 128 is designed to be used with the 1541 or 1571 disk drive. Compiled programs may be copied to any medium that will store a BASIC program and the compiler itself will work with other disk drives via a transparent serial interface. The following notes are provided to alert the user to a serious potential problem with the 1571 disk drive.

Problems with the 1571

The 1571 disk drive has been supplied by Commodore in a number of different versions. Unfortunately some versions contain a serious bug which affects Petspeed and it is not always possible to tell which version you have. The bug affects large sequential files and causes data to be lost in some circumstances. If you experience unexplained problems during compilation, particularly during pass 3 or you get a PARSE TREE CORRUPT ERROR you have probably encountered the problem. Here's what to do:

The fix for this problem is to use the 1571 in 1541 mode. To allow upward compatibility with the earlier 1541 disk drive, the 1571 can be told to go into "1541 mode". In this mode it behaves exactly like a 1541. Petspeed itself is distributed on a 1541 formatted disk so that it will work with either disk drive.

Using the 1571 in 1541 mode will not affect compilation in any way, neither will it prevent you from compiling any program that you could have compiled on the 1571. Moreover, as soon as your program is compiled you can copy it to a 1571 formatted disk simply by loading it into memory and saving it again.

When the 1571 first reads a disk, it checks to see whether it is 1541 or 1571 format. If the disk is 1541 formatted, the 1571 flashes its drive light a couple of times and then turns itself into a 1541 for as long as that disk is mounted. Compiling a program in 1541 mode is therefore simply a matter of providing the program to be compiled on a 1541 formatted disk. This is a piece of cake:

1 Place a disk which you don't mind being wiped clean in the 1571.

2 Put the computer in C64 mode by hitting the reset button while holding the Commodore key down. Do not type GO64 as this does not have the same effect. The 128 should now reset in 64 mode. Now format the disk using C64 speak, ie

```
OPEN 1,8,15,"n0:xxxx,yy":close 1
```

where xxxx is any name and yy is any 2 characters. The 1571 should now format the disk as a 1541 disk.

3 Hit the reset button again to bring the computer back into 128 mode.

From now on you need take no further action. The disk just formatted will always be treated by the 1571 as a 1541 disk and you can swap between 1571 formatted disks and 1541 formatted disks quite freely.

To summarize, if you always use 1541 formatted disks to compile programs you will not run into the 1571 problem.

Petspeed And Machine Code

This section contains all the information needed by the advanced programmer to interface machine code routines to Petspeed 128. It is recognised that certain types of routine will not run fast enough even when compiled and will need to be written in machine code. Do not simply assume however that a routine which needed to be machine coded when the program was interpreted still needs to be in machine code when the program is compiled. If you can replace a machine code routine with BASIC and still obtain sufficient speed you will be doing yourself a favour.

The object program

The object program is the executable file that results from the compilation of a BASIC source program under Petspeed. In this section we discuss the format of the object program, its size and the differences between it and a regular BASIC program.

Petspeed works by converting BASIC into a compact and rapidly executable pseudo machine code called speedcode.

When you list a compiled program, all you see is a couple of lines of BASIC, the last statement of which is SYS(16452). When this SYS statement is executed control passes to a machine code program called the Run Time System (RTS). The job of the RTS is to interpret the speedcode generated from your program by Petspeed. Control remains with the p-machine all the time your program is running. The p-machine lives in memory between $4000 and $7000. Because it is a non re-locatable machine code program it can only execute at this address and therefore, Petspeed compiled programs always run with the bit map screen allocated. The call SYS(40783) ensures this and the only statement which in BASIC can de-allocate the bit map screen (GRAPHIC CLR) does not have this effect in a compiled program.

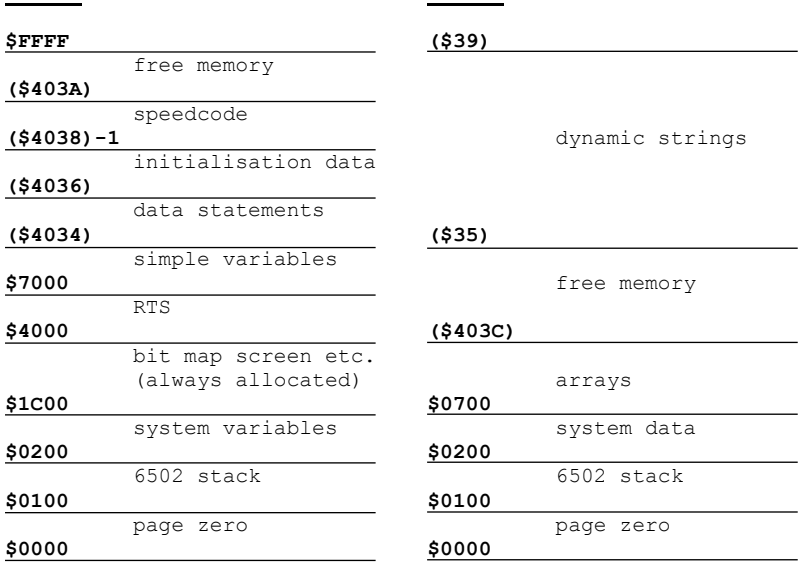
Object program size

It might appear that Petspeed's insistence upon having the bit map screen permanently allocated potentialy restricts the amount of memory that will be available for code. This in fact is not the case. Speedcode occupies significantly less space than BASIC and a program which would occupy the whole of bank 0 with the bit map screen unallocated is reduced in size sufficiently by Petspeed to make room for the screen.

One consequence of the p-machine overhead is that a compiled program can never be smaller than 45 blocks or about 11k. Small source programs will therefore produce much larger object programs. However, because speedcode is much more compact than BASIC text, there will come a point when this saving in space compensates for the size of the p-machine. At this point we come out even with the compiled program equal in size to the BASIC program. The break even point varies according to the type of program being compiled. Typically it is of the order of 80 blocks or 20k. Programs larger than this will become smaller when compiled.

The format of an object program

Fig 1 shows the memory maps of bank 0 and bank 1 for a compiled program. Broadly, the setup is similar to that of an ordinary BASIC program. There are some differences however the most important of which is that simple variables are held in bank 0 along with the code
and not in bank 1 as they are in regular BASIC.



NOTE hex addresses in parentheses refer to pointers. E.g. the start of data statements can be found by examining locations $4034 and $4035. All pointers reside in bank 0.

Calling machine code

Machine code calls are achieved from a compiled program exactly as they are from BASIC by mean of the SYS and BOOT statements or the USR function.

Many machine code programs will run without alteration from a compiled program. However there are two potential problems.

1 Memory Allocation

Petspeed increases the size of small programs and reduces the size of large ones. It is therefore possible for memory which was available for machine code when the program was interpreted to be used by the program once it is compiled. The areas of memory used by a compiled program can easily be ascertained from the segment pointers (see The Object Program) and in the worst case machine language programs may have to be re-assembled.

2 Variable Access

The variables in a compiled program differ in several ways from those created by a BASIC program. The differences are as follows.

a Simple variables (not arrays) are included in the object file and are held in bank 0; not in bank 1.

b Variables created by a compiled program do not have their names held along with the data.

c Numeric variables are stored in a different format from BASIC.

d Variables and array elements which are not declared as integers using the % sign will not necessarily be held in floating point format. The type of the data in an ordinary real variable or array element can change frequently at run time. The way to transfer values from Petspeed variables to machine code routines is to make then integer explicitly (e.g. A%, PQ%(10,2) etc). If this is not possible, a machine code routine can if necessary obtain the current type of a variable by looking at the type descriptor (see below). Should the programmer wish to pass a value back to Petspeed via a real/int variable, he may do so by forcing the type descriptor to the value appropriate for the type of data being passed.

Variable formats

The diagram below gives the format of all Petspeed data types. Corresponding diagrams illustrating the format of BASIC 7.0 variables are to be found in Commodore's own documentation

Simple Variables (Bytes marked X are not used)

Type	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
Real/Integer	EXP	M1	M2/MSB	M3/LSB	M4	SGN	0/1
Integer	X	X	MSB	LSB	X	X	5
String	X	X	LENGTH	PTR-LSB	PTR-MSB	X	-ve

The current type of real/int variables is specified by B7. The 32 most frequently accessed variables are treated specially and contain an extra unused byte (B8).

Arrays

Type	B1	B2	B3	B4	B5
Real/Integer	EXP/0	M1 & SGN/x	M2/MSB	M3/LSB	M4/x
Integer	MSB	LSB	(2 bytes used)		
String	LENGTH	PTR-LSB	PTR-MSB	(3 bytes used)	

The individual elements of real/int arrays can also change type. If the exponent is zero, the type is integer, otherwise the type is real.

A real/int variable is any numeric variable declared without a % suffix.

User Defined Functions

	B1	B2
Function defn block	PTR-LSB	PTR-MSB

PTR points to a Speedcode routine in bank 0 for evaluating the function. The pointer itself resides in bank 1.

The Report utility program

The report program which is supplied on the Petspeed system disk is a utility designed to give the user all the information he needs concerning the location of objects in the object file. The program displays a list of all variables, arrays and user defined functions together with their addresses. Also displayed are the addresses of the main program segments (see memory maps). To run report, proceed as follows.

1 Mount the system disk and type RUN "REPORT".

2 When requested, mount the work disk containing the SOURCE program and hit any key.

When REPORT has finished all the information can be re-displayed or printed by menu selection.

CAUTION any changes you make to your program will probably invalidate the information given by REPORT. Whenever you re-compile the program you should re-report as well.

Work files generated by Petspeed

In addition to the loadable object program (filename.BRON), Petspeed generates 6 work files on the work disk. Usually, all but one of these is scratched at the end of pass4 but if compilation is aborted for any reason they will remain on the directory. If the same program is recompiled however the new files will overwrite the old ones and these will be scratched in the normal way. Since the information contained in these files might conceivably be useful to the advanced programmer their contents are detailed below.

filename.a contains the array table, a list of the addresses of all string and integer variables and user defined functions together with certain miscellaneous data.

filename.d contains processed DATA statements. The contents of this file often bears little resemblance to the original DATA. The size of this file is proportional to the amount of data in the program.

filename.g contains a list of all GOTO, GOSUB, RESTORE, TRAP and RESUME reference addresses, the line number at which they occur and the target line number. The size of this file is proportional to the number of statement references in the program.

filename.t contains the parse tree. This is a large file often running into several hundred blocks. It contains a very detailed representation of the program in a form suitable for processing by the optimizer.

filename.er This file is not scratched by the compiler as are the other work files. It contains a list of all line numbers together with their addresses in the speedcode and is used by the ERRORS utility to locate run time faults.

filename.z This file is like the .w file but contains only the numbers of lines that contain DATA statements together with offsets into the .d file.

Terminology

There is a certain amount of jargon which is used frequently when compilers are under discussion. 128 users may not be familiar with this and the main terms are defined here.

compiler A compiler is a program for translating source text into a form suitable for direct execution. Sometimes the output of a compiler will be machine code but often it will be an intermediate PSEUDO CODE or P-CODE. On a microcomputer, a machine code generating compiler is unsuitable because of the very large object programs that would be produced (often two to three times the size of the source program).

interpreter This is a program which can read source text, decide what the programmer intends, and carry it out all at the same time. It is very convenient for debugging because it is the source program itself that is being run and the user can find errors and correct mistakes very easily. Because they have to do so much work however, interpreters are very slow and when a program is working properly, compilation will give improved performance.

source program This is the original program written (for our purposes) in BASIC 7.0.

object program This is the final output of a compiler. Petspeed object programs are directly loadable and executable.

speedcode This is the name of the type of code that Petspeed generates from the source program. Speedcode is a type of p-code and is designed to be compact and to execute rapidly.

compile time The time at which the program is compiled.

run time The time at which the program is run.

pass A pass occurs each time the compiler reads the source program or some representation of it. Petspeed is a four pass compiler.

optimisation Many compilers merely translate one language into another. Some compilers however, including Petspeed analyze the source program and attempt to improve its efficiency. BASIC is an inefficient language and can be much improved by optimisation.

end user An extremely nice chap.