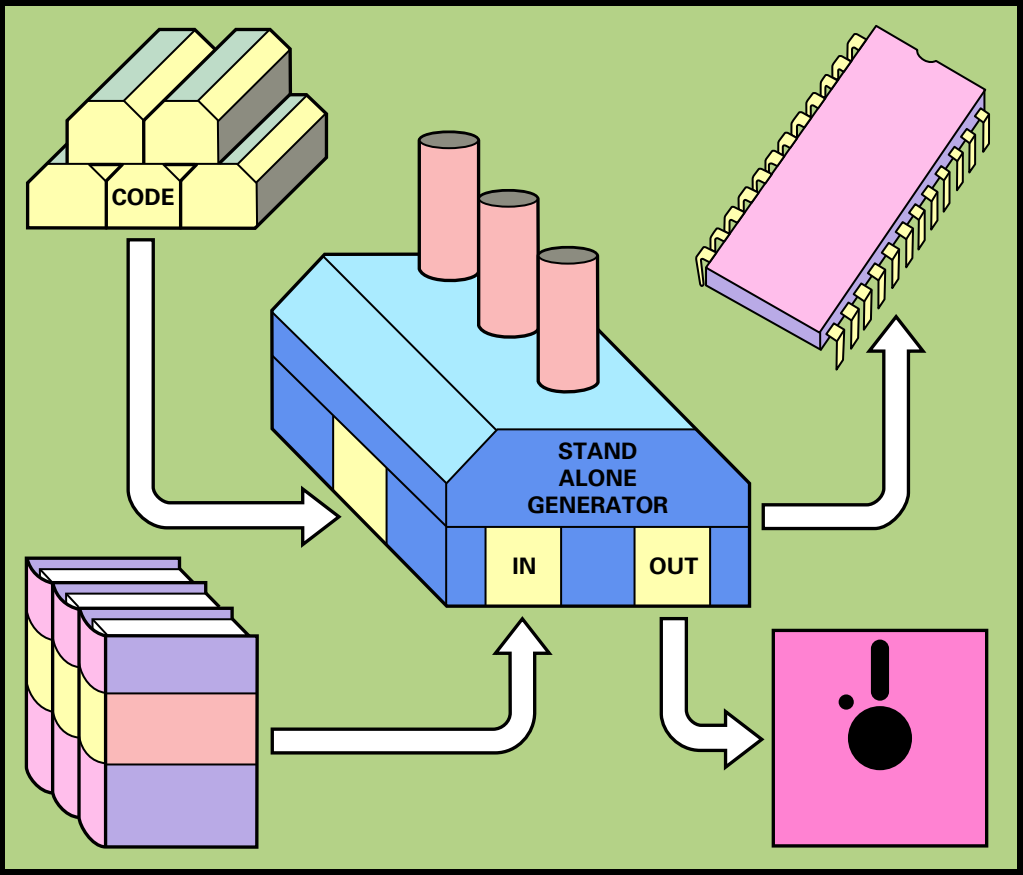
BCPL

User Guide

Stand Alone Generator

on the BBC Microcomputer

CHRIS JOBSON



User Guide

BCPL

Stand Alone Generator

on the BBC Microcomputer

CHRIS JOBSON

ACORNSOFT

Disclaimer

Richards Computer Products Ltd. and Acornsoft Ltd. reserve the right to make improvements in the product described in this book at any time and without notice. Every effort has been made to ensure the accuracy of the material presented in this book. Nevertheless, experience shows that some textual and software errors always remain to be discovered. If you find any errors, or have any suggestions on how to improve this book, please contact Richards Computer Products Limited, Brookside, Westbrook Street, Blewbury, Didcot, Oxfordshire OX11 9QA, England.

First published in 1983, by Acornsoft Limited

FIRST EDITION

Re-mastered by dv8 in 2020
First revision, November 2020

For the latest revision of this book go to:
stardot.org.uk/forums/viewforum.php?f=42

Published by:

Acornsoft Limited
Betjeman House
104 Hills Road
Cambridge
CB2 1LQ

Contents

1 Introduction 1
How to use this guide 2
System features 3
Requirements 5

2 Generating stand alone programs 7
Example command files 9

3 Stand alone environment 13
Starting and stopping 13
Differences from development environment 15
I/O options 19
Layout of store 22

4 Utilities 25
FILETRN - file transfer 26
FIXCIN - create program to run in RAM 27
PACKCIN - pack CINTCODE 31
ROMCIN - create program as language ROM 33
UTILCIN - create program as utility ROM 37

5 Procedures 41

6 Discussion 47
ESCAPE handling 48
Handling '*' commands 50
Location of global vector and heap 51
Machine code programs 53
Overlaying 57
Program termination 60
Programs to run on non-BBC machines 63
Replacing SYSLIB procedures 65
Writing programs to execute in ROM 65

7 Summaries 69
Error numbers and fatal error codes 70
Files provided 71
Global variables 72
Global vector 74
Utilities 74
Utility error messages 75

Appendix 79
Globals used by Stand Alone Generator 80
Re-use of global numbers 80
Running in the 6502 second processor 81
Sections in SYSLIB 82
Sizes 86
Stand alone file formats 88
System data areas 94
System procedures 95

1 Introduction

The BCPL CINTCODE system on the BBC Microcomputer provides a powerful environment for the development of BCPL and small assembler programs, but programs developed using it can normally only be run on BBC Microcomputers fitted with the BCPL Language ROM.

The BCPL Stand Alone Generator greatly enhances the usefulness of the BCPL CINTCODE system by permitting the easy distribution of developed BCPL programs. The utilities in the package convert developed programs into stand alone programs which can be run on any BBC Microcomputer, whether or not it is fitted with the BCPL Language ROM. Indeed it is even possible to develop stand alone programs for other 6502-based computers (eg for dedicated control applications).

Stand alone programs can be generated in two main formats:

- as files which may be held on any suitable medium and which are run using the '*RUN filename' filing system command (often this can be shortened to '*filename');

- as language ROMs which can be plugged into the sideways ROM sockets and which are run by '*name' commands (the names used are chosen when the stand alone program is generated).

The Stand Alone Generator is designed for use with the BCPL CINTCODE system on the BBC Microcomputer. It works by linking the application program with the BCPL interpreter and an appropriate subset of the BCPL library procedures.

Some features of the development system (ie the BCPL CINTCODE system) are not available to stand alone programs. In particular store files and the separate run and command states are not supported. Nevertheless, converting programs to run in a stand alone environment is generally straightforward and often very few, if any, modifications will be required.

Licensing arrangements

All BCPL programs produced by the Stand Alone Generator include the BCPL interpreter and a number of BCPL library procedures. The interpreter and library procedures are copyright software and a licence is needed to distribute programs containing them.

A licensing agreement is included with the Stand Alone Generator and its terms should be studied before distributing any stand alone programs.

Special Requirements

The Stand Alone Generator is designed to cope with the majority of foreseeable requirements for creating stand alone BCPL and assembler programs. Any users with special requirements involving changes or additions to the Stand Alone Generator software are invited to contact Richards Computer Products Limited for assistance.

HOW TO USE THIS GUIDE

All readers are recommended to read the remainder of this Introduction as well as chapters 2 and 3, which give an overview of the use of the package and of the environment in which stand alone programs run.

Readers preparing to generate a stand alone program should follow the steps outlined in chapter 2, referring as necessary to chapters 4, 5 and 6 which respectively explain how to use the utilities, summarise the library procedures available and discuss various features of stand alone programs.

Chapter 7 contains summaries of the utilities and the stand alone environment. It may be used for quick reference by experienced users.

The Appendix describes the various features provided and contains other information which may be of interest to experienced users or to users with special requirements.

This guide assumes that the reader is familiar with the BCPL CINTCODE development system and has access to the User Guide for that system. The sections covering language ROMs assume some familiarity with the concepts of sideways ROMs, but very little knowledge of this subject is needed to generate a stand alone program as a language ROM using this package.

SYSTEM FEATURES

Language ROMs and utility ROMs

The Stand Alone Generator has the ability to create two types of sideways ROMs, referred to in this guide as **language ROMs** and **utility ROMs**, (although both types are actually language ROMs from the point of view of the BBC Microcomputer operating system).

The difference is that language ROMs have only one entry point whereas utility ROMs have several. Typically a language ROM would be used for a large stand alone program with one basic function (eg an editor), whereas a utility ROM could be used to hold a number of small programs, each entered by its own '*name' command (eg a set of disc utilities).

Utility programs

The Stand Alone Generator contains five utility programs:

FILETRN is a utility to copy a file preserving the load and execution addresses. It can copy files from one filing system to another (eg from disc to tape) or from one disc to another even if only one disc drive is available.

FIXCIN is the utility which creates a stand alone program as a file which can be run by the '*RUN' command. It also creates stand alone programs for target systems other than the BBC Microcomputer.

PACKCIN is a utility to compact CINTCODE files by removing SECTION and NEEDS directives and merging hunks. It is most useful for minimising the space required by stand alone programs but can also be used to advantage on programs to be run in the development environment.

ROMCIN is the utility which creates a stand alone program as a language ROM.

UTILCIN is the utility which creates a stand alone program as a utility ROM.

System libraries

The system library files supplied with the Stand Alone Generator contain the CINTCODE of special versions of the library procedures in the BCPL Language ROM and the development system library, LIB. They also contain a special version of the BCPL interpreter.

Two library files are provided, SYSLIB1 and SYSLIB2. They differ only in some of the I/O procedures. The term 'SYSLIB' is used throughout this guide as a generic term for these two files when the differences between them are unimportant.

REQUIREMENTS

A Model B BBC Microcomputer with the BCPL CINTCODE system and a filing system such as disc or Econet (cassette tape only is not sufficient) is required to use the Stand Alone Generator.

The output of the utilities ROMCIN and UTILCIN is a ROM image file. To copy this file into an EPROM, an EPROM programmer and suitable software to drive it is required.

The computer requirements of particular stand alone programs depend largely on the programs themselves, but to use ROMs generated by ROMCIN or UTILCIN operating system 1.0 or later is required and to access disc files DOS version 0.90 or later is required.

Many programs produced by the Stand Alone Generator will run unchanged in a 6502 second processor. The Appendix details the effects of running in the second processor.

2 Generating Stand Alone Programs

A BCPL CINTCODE program is converted into a stand alone program by linking the CINTCODE with the library routines used and with the interpreter (both these are extracted from SYSLIB). The resulting file is compacted using PACKCIN then converted by FIXCIN, ROMCIN or UTILCIN into a stand alone program file or a ROM image file.

The use of PACKCIN is not essential but results in a significantly smaller stand alone program.

In more detail the recommended sequence of events for producing a straightforward stand alone BCPL program is as follows:

(1) Write the program and test it (as far as possible) using the development system.

(2) Make any changes necessary to the code for the stand alone version.

(3) Go carefully through the program, listing all library routines used (this includes routines from the BCPL language ROM as well as those from LIB). Look up the entry in chapter 5 for each of these routines and thus prepare a list of the sections to be extracted from SYSLIB. (This process also checks that the program does not use any routines not available in the stand alone environment.)

(4) Add the section "INTERP" (which contains the interpreter) to this list.

(5) Decide which, if any, of the I/O options described in chapter 3 is required and add the appropriate section to the list.

(6) Decide which, if any, of the termination options described in 'Program termination' in chapter 6 is required and add the appropriate section to the list.

(7) Include the required NEEDS directives in the program source. This may be done either by including the relevant directives in individual source files, by including all the directives in one of the source files or by preparing a separate source file containing all the directives. The third option is recommended (the source file need not contain any code).

(8) Compile all the program source files (including the new one with the NEEDS directives) with the NONAMES option (this saves space in the stand alone program). Use JOINCIN to join all the CINTCODE files together.

(9) Decide whether the sections NEEDed should be extracted from SYSLIB1 or SYSLIB2. The difference is important only if the program uses file I/O. See 'I/O options' in chapter 3 for details.

(10) Use NEEDCIN to extract the sections from the chosen SYSLIB file. The result of this step is a CINTCODE file containing the program, the interpreter and all necessary library procedures.

(11) Use PACKCIN to compact the CINTCODE file into a smaller CINTCODE file. This removes the section names and NEEDS directives thus resulting in a smaller final program.

(12) If the stand alone program is to be built as a language or utility ROM then create a 'Spec' file, which contains details of the ROM header and entry points (see the sections on ROMCIN and UTILCIN in chapter 4).

(13) Decide on the location of the global vector and heap (see 'Location of global vector and heap' in chapter 6). Decide whether a particular screen mode should be set up before the stand alone program is entered. These decisions affect the parameters that will be given to FIXCIN, ROMCIN or UTILCIN.

(14) Use FIXCIN, ROMCIN or UTILCIN to create the stand alone program, either as a file which can be executed by '*RUN' (from FIXCIN) or as a ROM image file. In the case of ROM image files check that the file will fit in the type of EPROM it is proposed to use, remembering that the maximum size of a language ROM is 8192 words.

(15) If the output file from FIXCIN is to be copied then use FILETRN (rather than READ/SAVE/COPY) since the file contains load and execution addresses which must be preserved.

EXAMPLE COMMAND FILES

It is suggested that EX command files are set up to assist in the process of building stand alone programs. Two examples of typical command files are given here.

Example 1

This command file uses FIXCIN to create a stand alone program. The parameters to the command file are:

INFILE the name of the CINTCODE file containing the program;

OUTFILE the name to be given to the stand alone program file. (FIXCIN always creates the stand alone program file as a current filing system file.)

The command file assumes that SYSLIB1 is to be used, that the global vector is to be located at hex byte address 1900 and that all optional FIXCIN parameters are defaulted.

Store files are used for the temporary files needed. This works only for fairly small programs. For larger programs there is not enough room in store and the current filing system has to be used for these temporary files (see the second example).

```
.KEY INFILE/A,OUTFILE/A
NEEDCIN <INFILE> SYSLIB1 $TEMP1
PACKCIN FROM $TEMP1 TO $TEMP2
DELETE $TEMP1
FIXCIN FROM $TEMP2 TO <OUTFILE> GV=1900
DELETE $TEMP2
```

Example 2

This command file uses ROMCIN to create a ROM image file. The parameters to the command file are:

INFILE the name of the CINTCODE file containing the program;

OUTFILE the name to be given to the ROM image file. The command file creates this as a current filing system file;

SPEC the name of the 'Spec' file;

GV the hex byte address at which the global vector is to be located (defaults to 1900 if not specified);

LIBNO '1' or '2' for SYSLIB1 or SYSLIB2 (defaults to SYSLIB1 if not specified);

PARS any additional parameters, to be given to ROMCIN eg if the screen mode is to be set to 5 then the EX command line might include:

PARS = "MODE 5"

Filing system files are used for all temporary files to avoid running out of store when dealing with large programs.

```
.KEY INFILE/A,OUTFILE/A,SPEC/A,GV,LIBNO,PARS
.DEF GV=1900
.DEF LIBNO=1
NEEDCIN <INFILE> SYSLIB<LIBNO> /F.$TEMP1
PACKCIN $TEMP1 /F.$TEMP2
*DELETE $TEMP1
ROMCIN $TEMP2 /F.<OUTFILE> <SPEC> GV=<GV> <PARS>
*DELETE $TEMP2
```

3 Stand Alone Environment

The environment provided for stand alone BCPL programs can be considered as a much simplified form of the development environment. The principal differences from the development environment (which are discussed in more detail below) are:

- there is no command state;
- the main stack is of fixed size;
- there is no automatic ESCAPE handling;
- store files are not provided.

STARTING AND STOPPING

Starting

A stand alone program which has been created as a file is started by selecting the appropriate filing system and typing:

*RUN filename

(in some filing systems this can be abbreviated to *filename).

The file is loaded by the filing system into a pre-defined place in RAM and the initialisation code is entered. This code optionally selects a particular screen mode and sets up the global vector, the main stack and the heap. It then enters the interpreter which starts executing a CINTCODE initialisation routine.

The size of the global vector is governed by the maximum global used or defined in the program. Unused entries are initialised to GLOBWORD. The size of the main stack is fixed. The stack is not initialised to zeros.

The CINTCODE initialisation routine initialises various globals and I/O streams and then calls the application procedure START.

A stand alone program which has been created as a language or utility ROM is entered by typing:

*name

where **name** is one of the names associated with the ROM (these names are specified as input to ROMCIN or UTILCIN).

The initialisation process is similar to that described above, except that the code is all executed in ROM and therefore is not loaded into RAM. In the case of a utility ROM the first BCPL procedure called by the CINTCODE initialisation routine is not necessarily START but depends on the particular **name** used.

If the language or utility ROM is installed as the right-most language ROM in a BBC Microcomputer then it is entered automatically when the computer is switched on or if CTRL-BREAK is pressed.

Pressing BREAK when a stand alone program is running re-enters the current language ROM. If the stand alone program was created by FIXCIN then this re-enters the language that was running when the '*RUN' command was entered (in the 6502 second processor BREAK crashes the system and CTRL-BREAK must be used to break out of the program). If the stand alone program was created by ROMCIN or UTILCIN then this re-enters the stand alone program.

Stopping

All the usual methods of stopping are available to stand alone programs, ie returning from START (or whichever procedure was first entered in the case of programs generated by UTILCIN), calling ABORT, ENDPROG or STOP or executing the statement FINISH.

Various options are available to control what happens when a stand alone program stops and these are detailed in 'Program termination' in chapter 6. In brief the options available are:

- prompt the user to enter a '*' command for the language or utility he wants to use next;

- re-enter the current language (but this does not work for programs running in a 6502 second processor);

- do nothing (ie the machine will lock up until BREAK is pressed).

The first of these options is likely to be the best in many cases, but often it will be better for the program itself to incorporate a means of exiting without calling STOP etc. Such a means might be for the program to ask the user for a '*' command or for the program to always return to BASIC. Some programs may never exit (eg dedicated control systems).

Note that if a stand alone program is in the right-most language ROM in a BBC Microcomputer then the user has no way of entering '*' commands (eg to set up serial I/O baud rates or enter other language ROMs) unless either the stand alone program explicitly allows him to enter '*' commands or the stand alone program exits with the first option mentioned above.

DIFFERENCES FROM DEVELOPMENT ENVIRONMENT

This section describes a number of the differences between the stand alone environment and the development environment, and suggests the types of program changes that may be necessary to cope with these differences.

Argument handling

When a stand alone program is run (by '*RUN file' or '*name') it is not possible to read the program's arguments from the command line. Thus a program that expects arguments should prompt for them explicitly before calling RDARGS.

A good way to do this might be to use the string passed to RDARGS (in the same way as RDARGS itself does if the user enters '?') eg

```
ARGSTR := "FILE/A,NEW"
WRITES(ARGSTR); WRITES(":*N")
IF RDARGS(ARGSTR, ARGVEC, SIZE) = 0 THEN ...
```

Command state and ESCAPE

In the development system the existence of a run state and a command state allows application programs to trap to the command state in certain error conditions (eg running out of store) and also allows the user to interrupt application programs.

In the stand alone environment there is only one state and by default the ESCAPE key is disabled. There are a number of significant effects:

- the TRAP procedure cannot be used (as there is nowhere to trap to).

- the program must check all GETVEC calls for success and must itself handle the consequences of a GETVEC call failing.

- ESCAPE cannot be used to abandon cassette tape I/O.

- if the user is to be given the facility to interrupt the program then the program itself must check at intervals to see if the user has interrupted (by whatever method is chosen) and must itself take whatever action is required. 'ESCAPE handling' in chapter 6 discusses the effects of the program re-enabling ESCAPE.

- the program itself may have to provide a way for users to enter the '*' commands which they can enter in the command state in the development environment (eg to look at a disc directory or specify the printer ignore character). 'Handling '*' commands' in chapter 6 discusses this subject in more detail.

Coroutines

Coroutines can be used in the same way as they are used in the development system. One minor difference is that in the stand alone system the main stack is not linked into COLIST.

Error checking

Many of the error checks made in the library procedures in the development system are omitted in the stand alone versions of these procedures (eg VECTOFILE does not check that the vector is a heap vector).

Thus it is important to debug programs fully before converting them to stand alone programs. It is also important that programs do not rely too much on the error checking in the library procedures (eg by deliberately causing errors and then trapping the resultant ABORTs). Chapter 7 lists the error codes which are supported in the stand alone system.

RUNPROG of built-in commands and utilities

Although the procedure RUNPROG is available in the stand alone system, its only function is to issue operating system/filing system commands. Thus it cannot be used to run 'built-in' commands such as PAUSE or utilities such as JOIN.

STARTINIT and stack size

In the development system the procedure STARTINIT is used to enable a program to set up its environment before START is called, and in particular to say what size stack it needs.

In the stand alone system STARTINIT is completely ignored and all stand alone programs are entered with approximately 440 words of stack available. If this is insufficient then two options are available:

- reduce the stack requirement (eg by replacing VECs by calls to GETVEC);

- treat the main body of the program as a co-routine and call CREATECO (to set up a stack of the required size) then CALLCO (to enter the main body of the program) from START.

Store files

The stand alone environment does not support store files and does not recognise the device '/S'. In particular procedures such as FINDOUTPUT and DELFILE, which in the development system treat a file name without a device specifier as a store file name, in the stand alone system treat it as a current filing system file name.

The procedures FILETOVEC, READVEC, SAVEVEC and VECTOFILE are all available however.

I/O OPTIONS

Support of the full range of I/O devices provided in the development system requires a significant amount of code. In order to leave more space free for applications code which uses only a subset of the full I/O facilities a number of different I/O options are provided.

A particular option is selected by including a particular NEEDS directive in the stand alone program. Only one of the I/O options may be included in any one program.

Note that use of FILETOVEC, READVEC, SAVEVEC and VECTOFILE is independent of the choice of I/O option (with one exception discussed in 'Choice of SYSLIB1 or SYSLIB2' below). Thus I/O option 2 (which only provides console and keyboard I/O) may be selected for a program which writes files away to the current filing system using SAVEVEC.

The options available are:

Default

This option is selected if none of the NEEDS directives for other options are included. The only I/O procedures available are SELECTOUTPUT, WRBIN, WRCH and WRITES.

SELECTOUTPUT is a dummy procedure which has no effect.

WRBIN, WRCH and WRITES all output to the current OSWRCH destination. On entry to the program this is the screen.

Since no stream input is provided, this option is most likely to be useful for a program which performs its own I/O using the appropriate operating system calls.

Option 1

This is selected by NEEDS "INOUT1" and provides one input stream which is the console read a line at a time with echo (equivalent to /C in the development system) and one output stream which is the screen. CNSLINSTR and CNSLOUTSTR point to these streams.

The I/O procedures available are ENDREAD, ENDWRITE, FINDINPUT, FINDOUTPUT, INPUT, OUTPUT, RDBIN, RDCH, SELECTINPUT, SELECTOUTPUT, UNRDCH, WRBIN, WRCH and WRITES.

ENDREAD, ENDWRITE, SELECTINPUT and SELECTOUTPUT are all dummy procedures. FINDINPUT and FINDOUTPUT are also dummy procedures which return pointers to the one input stream and the one output stream respectively.

Input is performed by calling OSRDCH and output by calling OSWRCH. On entry to the program the OSRDCH source is the keyboard and the OSWRCH destination is the screen. If these assignments are changed (eg by *FX2 or *FX3 calls) then the BCPL I/O will use the new assignments.

Option 2

This is selected by NEEDS "INOUT2". It is identical to option 1 except that the input stream is the keyboard read a character at a time with no echo (equivalent to /K in the development system).

Option 3

This is selected by NEEDS "INOUT3". It provides all the I/O devices available in the development system apart from store files and the current filing system ie /C, /E (errorstream), /K, /L, /N and /P.

The list of procedures available is the same as for option 1, but none of them are dummies.

On entry to the program the current input stream is CNSLINSTR (/C) and the current output stream is CNSLOUTSTR (/C). ERRORSTREAM points to an error stream, as in the development system.

Option 4

This is selected by NEEDS "INOUT4". It is similar to option 3 but also supports the current filing system. Names beginning '/F.' or not beginning with '/' are taken as current filing system file names.

One additional procedure, FSTYPE, is provided by this option.

There are two different versions of the section INOUT4, one in SYSLIB1 and the other in SYSLIB2. The differences are detailed below.

Choice of SYSLIB1 or SYSLIB2

The differences between SYSLIB1 and SYSLIB2 are irrelevant except when using I/O option 4 or any of the procedures FILETOVEC, READVEC, SAVEVEC and VECTOFILE.

The SYSLIB1 procedures assume that the current filing system is either disc, Econet or some similar filing system ie is a filing system whose FSTYPE attributes are 'supports OSGBPB' and 'OSFILE can read file length'. The version of FSTYPE provided by SYSLIB1 for I/O option 4 always returns these attributes, irrespective of the true characteristics of the current filing system.

An attempt to access a tape or ROM file from a program using SYSLIB1 procedures will probably fail in an obscure fashion.

The SYSLIB2 procedures adapt to the characteristics of the current filing system in the same way as the I/O procedures in the development system.

An important restriction when using SYSLIB2 is that if FILETOVEC or READVEC are used then I/O option 4 is automatically selected. If the program attempts to use a different I/O option as well then the results are unpredictable.

LAYOUT OF STORE

There are three main areas used by stand alone BCPL programs - zero page, the language RAM and the free RAM from the operating system/filing system workspace up to the display RAM.

Zero page

The use made of this area is identical to the use made of it by the development system, except that some of the data areas used by the development system are not used by stand alone programs. Details are given in the Appendix. Bytes 112 to 143 are still available for applications use.

Language RAM

This area contains some system data (again very similar to that used by the development system) and the main stack. In the stand alone system the main stack is not part of the heap.

Free RAM

The free RAM contains the global vector, the heap and, in the case of programs generated by FIXCIN, the program itself. Parameters to FIXCIN, ROMCIN and UTILCIN can specify the addresses of each of these areas individually if required, but defaults are provided as follows.

In all cases the base address of the global vector has to be specified. This will normally be chosen to be the lowest address not used by the operating system in the target machine (eg for a Model B with discs this address is byte 1900 hex).

The global vector is positioned at the specified address. Its size is governed by the highest global number defined in, or referenced by, the program.

For programs generated by FIXCIN the program immediately follows the global vector. The program proper is followed by initialisation code. Once initialisation is complete this code is discarded and becomes part of the heap, which runs from the top of the program proper to the bottom of display memory.

For programs generated by ROMCIN or UTILCIN the heap runs from the top of the global vector to the bottom of display memory.

'Location of global vector and heap' in chapter 6 discusses various alternatives to the default options. The Appendix explains the layout of store when running in a 6502 second processor.

4 Utilities

This chapter describes the utility programs provided with the Stand Alone Generator. These are as follows:

FILETRN copies a filing system file, preserving the load and execution addresses. It can copy files from one filing system to another.

FIXCIN creates a stand alone program as a file that can be executed by the '*RUN' command.

PACKCIN compacts a CINTCODE file by removing SECTION and NEEDS directives and merging hunks.

ROMCIN creates a stand alone program as a language ROM.

UTILCIN creates a stand alone program as a utility ROM.

Many of the error messages produced by the utilities are self-explanatory. Those that need more explanation are listed in 'Utility error messages' in chapter 7.

FILETRN - file transfer

Purpose

To copy a filing system file, preserving the load and execution addresses, to another file in either the same filing system or a different filing system. FILETRN is particularly useful for copying the stand alone program files produced by FIXCIN.

Examples

```
(1) FILETRN :0.MYPROG :1.MYPROG
(2) FILETRN MYPROG MYPROG PAUSE
(3) FILETRN MYPROG MYPROG *DISC *TAPE
```

Arguments

FROM/A,TO/A,FROMFS,TOFS,PAUSE/S

FROM The name of the file to be copied. The file must be a filing system file. The name should be the 'pure' file name ie it should not include '/F.'.

TO The file name of the copy. Again this must be a 'pure' file name. The file is created as a filing system file.

FROMFS The filing system containing the file to be copied. If not specified the current filing system is used. If specified it must be the '*' command used to select the filing system (see example 3 above).

TOFS The filing system in which the copy is to be created. If not specified the filing system from which the file was copied is used. If specified it must be the '*' command used to select the filing system (see example 3 above).

PAUSE If specified the program will pause (with the message 'Type CONT to resume') after reading the **FROM** file but before writing the **TO** file. This allows a file to be copied from one disc to another on a system with only a single disc drive.

Remarks

FROMFS must be one which supports the OSFILE call to read the length of a file. Thus a file cannot be copied from tape using this program (though it can be copied to tape).

The program works by reading the file as a contiguous store file. Thus the biggest file that can be handled is dependent on the amount of contiguous free heap space available when the program is run.

If a store file exists with the same name as **FROM** then it is deleted.

If **TOFS** is specified then it will be the current filing system when the program exits.

FIXCIN - create program to run in RAM

Purpose

To generate a stand alone program as a file that can be executed by the '*RUN' command. This program can also be used to generate stand alone programs to run in target hardware other than the BBC Microcomputer.

Examples

```
(1) FIXCIN MYPROG MYFILE GV=1900
(2) FIXCIN MYPROG MYFILE GV=16AF MODE 5 MAX
    REPORT /L
```

Arguments

FROM/A,TO/A,GV/A/K,BASE/K,HEAP/K,HEAPEND/K,MODE/K,
MAX/S,NOTBBC/S,REPORT/K,NOINT/S

FROM The name of the CINTCODE file containing the program to be made stand alone.

TO The name of the stand alone file to be created. This must be a 'pure' file name (ie with no device specifier) and is taken to be a current filing system file.

GV The hex byte address of the base of the global vector in the stand alone system.

BASE The hex byte address of the program in the stand alone system. If not specified the program follows the global vector. This address is the load address and the execution address of the **TO** file.

HEAP The hex byte address of the heap in the stand alone system. If not specified the heap follows the program. The address is rounded up to a 4-byte boundary.

HEAPEND The hex byte address of the end of the heap in the stand alone system. If not specified the heap extends to the base of the display RAM. The address is rounded down to a 4-byte boundary.

MODE The screen mode to be set up before the stand alone program is entered (a number in the range 0 to 7). If not specified the screen mode is left unchanged when the program is entered.

MAX By default the stand alone program file is built up in RAM then copied out to the current filing system when complete. When building large stand alone programs this strategy may fail through running out of RAM. Specifying **MAX** causes the stand alone program file to be written to the current filing system as it is built up, allowing large programs to be created at the expense of FIXCIN taking longer to run.

NOTBBC This is specified when building a stand alone program for a target machine other than the BBC Microcomputer. In this case **BASE**, **HEAP** and **HEAPEND** must all be specified but **MODE** may not be specified. See 'Programs to run on non-BBC machines' in chapter 6 for more details.

REPORT When FIXCIN finishes it normally displays a report on the screen giving details of the file and stand alone environment produced. The **REPORT** argument allows this report to be written to another device eg a disc file or the printer (as in example 2).

NOINT If the stand alone program is written entirely in assembler and contains no BCPL code at all (and therefore the BCPL interpreter has not been included in the **FROM** file) then this argument must be specified. See 'Machine code programs' in chapter 6 for more details.

Remarks

The file FIXINI (supplied as part of the stand alone generator) is read by FIXCIN, and must therefore either be in store or on the current filing system device when FIXCIN is run.

Execution of the application code in **FROM** begins at the procedure START.

Unless the **MAX** argument is specified any store file with the same name as the **TO** file will be deleted, even though the **TO** file is created as a filing system file.

The report produced by FIXCIN gives the following details (in both decimal words and hex bytes):

- the size of the **TO** file;

- the base address and size of the global vector;

- the base address and size of the code (ie program). The base address of the code is also the load address and execution address of the **TO** file. The difference between the size of the code and the size of the file is the size of the initialisation code and data (which is normally overwritten by the heap once initialisation is complete).

- the addresses of the heap and (if **HEAPEND** has been specified) of the end of the heap. If **NOINT** has been specified then this information is omitted as no heap will be set up.

Since the **TO** file contains load and execution addresses it should be copied with FILETRN rather than with READ, SAVE, COPY etc.

PACKCIN - pack CINTCODE

Purpose

To compact a CINTCODE file by removing SECTION and NEEDS directives and merging hunks. PACKCIN is primarily intended for use with the Stand Alone Generator, since it is often important to minimise the size of stand alone programs, but since its output is a valid CINTCODE file it can also be used to reduce the size of programs run in the development system.

Examples

(1) PACKCIN CODEFIL PACKFIL
(2) PACKCIN MYFILE /F.P.MYFILE REPORT /L

Arguments

FROM/A,TO/A,REPORT/K

FROM The name of the input file ie the CINTCODE file which is to be compacted.

TO The name of the output file.

REPORT When PACKCIN finishes it normally displays a report on the screen giving details of the file created. The **REPORT** argument allows this report to be written to another device eg a disc file or the printer (as in example 2).

Remarks

The **TO** file is created from the **FROM** file by discarding all SECTION names and NEEDS directives, as well as the name of the first procedure in each hunk that contains procedure names. The hunks in the **TO** file are made up of the hunks from the **FROM** file according to the following rules:

- any hunks consisting only of SECTION and NEEDS directives are discarded;

- all machine code hunks are copied before any BCPL hunks (but the ordering of hunks is otherwise maintained);

- as many hunks as possible are merged subject to the restriction that no output hunk can contain more than 4095 words of code (except in the case of an input hunk bigger than this which is copied unchanged);

- the type of an output hunk is 'machine code' if it is made up from either all machine code hunks or from a mixture of machine code and BCPL hunks. The type is 'BCPL' otherwise.

If the **FROM** file contains more than one definition of the same global then the **TO** file contains only the last such definition.

The report produced by PACKCIN shows (in both decimal words and hex bytes):

- the size of the **TO** file;

- the size of the code in the **TO** file ie excluding hunk headers, global definitions and machine code relocation data;

- the space in the **TO** file taken up by global definitions.

FIXCIN, ROMCIN and UTILCIN discard hunk headers and relocate machine code when creating a stand alone program. In addition FIXCIN incorporates the global definitions into the initialisation data that normally becomes part of the heap. Thus the last two entries in the report are useful for estimating the space that will be used by **TO** if it is converted to a stand alone program.

ROMCIN - create program as language ROM

Purpose

To create a stand alone program as a ROM image file of a BBC Microcomputer language ROM.

Examples

```
(1) ROMCIN MYPROG MYROM MYSPEC GV=1900
(2) ROMCIN MYPROG /F.MYROM :1.MYSPEC GV 1900
    HEAP 2FAA REPORT /L
```

Arguments

FROM/A,TO/A,SPEC/A,GV/A/K,HEAP/K,HEAPEND/K,MODE/K, REPORT/K,NOINT/S

FROM The name of the CINTCODE file containing the program to be made stand alone.

TO The name of the ROM image file to be created. This is a normal file name and so will be a store file unless a device specifier is included.

SPEC The name of the 'Spec' file. This is a file containing details of the ROM header and the '*' commands to enter the ROM. See 'The Spec file' below for more details.

GV The hex byte address of the base of the global vector in the stand alone system.

HEAP The hex byte address of the heap in the stand alone system. If not specified the heap follows the global vector. The address is rounded up to a 4-byte boundary.

HEAPEND The hex byte address of the end of the heap in the stand alone system. If not specified the heap extends to the base of the display RAM. The address is rounded down to a 4-byte boundary.

MODE The screen mode to be set up before the stand alone program is entered (a number in the range 0 to 7). If not specified the screen mode is left unchanged when the program is entered.

REPORT When ROMCIN finishes it normally displays a report on the screen giving details of the file and stand alone environment produced. The **REPORT** argument allows this report to be written to another device eg a disc file or the printer (as in example 2).

NOINT If the stand alone program is written entirely in assembler and contains no BCPL code at all (and therefore the BCPL interpreter has not been included in the **FROM** file) then this argument must be specified. See 'Machine code programs' in chapter 6 for more details.

The Spec file

ROM header

The format of the ROM header (ie the first few words) of a language ROM is defined by the BBC Microcomputer Operating System. Although the user of the Stand Alone Generator need not be concerned with the details of this format, four items within the header must be specified.

These items are the version number (a decimal number in the range 0-255 used to identify different releases of the same ROM), the version text (not necessarily related to the version number), the copyright text (detailing the copyright in the ROM) and the title (which is displayed by the operating system when the ROM is entered).

Although the title is displayed when the ROM is entered, it is immediately erased if the **MODE** argument is specified (changing mode clears the screen).

The copyright text is never displayed but must begin with '(C)'.

Spec file format

The Spec file is a text file (normally it would be created with ED or TED) which specifies the four items listed above and the '*' command(s) that will cause the ROM to be entered.

All lines that do not begin with '.' are ignored and may be used for commenting the file. All other lines must begin with one of the five commands '.ENTRY', '.VERNO', '.VERSTR', '.TITLE' and '.COPYR'.

The commands may be specified in any order. The only one that is mandatory is .ENTRY, which specifies a '*' command that will cause the program to be entered.

The first item on the line after .ENTRY is taken as the '*' command. It need not include the '*'. The rest of the line may be used for comments. When a '*' command is entered upper and lower case are considered equivalent. Thus the '*' command

```
*fREd
```

would enter a ROM whose Spec file included

```
.ENTRY FRed
```

The Spec file may contain more than one .ENTRY command, allowing different '*' commands to be accepted - see the example below.

The version number is initially set to 0. It may be changed by the .VERNO command, whose parameter is a decimal number in the range 0-255.

The version text and title are both initialised to spaces. They may be changed by the .VERSTR and .TITLE commands respectively. In both cases the rest of the line (from the first non-space character after .VERSTR or .TITLE) is taken as the text.

The copyright text is initially set to '(C)'. It may be changed by the .COPYR command. The rest of the line (from the first non-space character after .COPYR) is taken as the copyright text. The first three characters must be '(C)'.

Example

```
Spec file for editor produced by XYZ Ltd
.VERNO  3
.VERSTR 2.13
.TITLE  XYZEDIT
.COPYR  (C) 1983 XYZ LTD.
.ENTRY  XYZEDIT full form
.ENTRY  XYZE.   abbreviation
.ENTRY  EDXYZ   yet another version
```

Remarks

The file ROMINI (supplied as part of the Stand Alone Generator) is read by ROMCIN, and must therefore either be in store or on the current filing system device when ROMCIN is run.

Execution of the application code in **FROM** begins at the procedure START.

The report produced by ROMCIN gives the following details (in both decimal words and hex bytes):

- the size of the **TO** file (ie the size of the ROM image);

- the base address and size of the global vector;

- the addresses of the heap and (if **HEAPEND** has been specified) of the end of the heap. If **NOINT** has been specified then this information is omitted as no heap will be set up.

UTILCIN - create program as utility ROM

Purpose

To create a stand alone program as a ROM image file of a utility ROM ie a BBC Microcomputer language ROM which has multiple entry points.

Examples

```
(1) UTILCIN MYPROG MYROM MYSPEC GV=1900
(2) UTILCIN MYPROG /F.MYROM :1.MYSPEC GV=1900
    HEAP=2300 HEAPEND=3000 REPORT=/F.MYREPORT
```

Arguments

FROM/A,TO/A,SPEC/A,GV/A/K,HEAP/K,HEAPEND/K,
REPORT/K

FROM The name of the CINTCODE file containing the program to be made stand alone.

TO The name of the ROM image file to be created. This is a normal file name and so will be a store file unless a device specifier is included.

SPEC The name of the ‘Spec’ file. This is a file containing details of the ROM header and the ‘*’ commands to enter the ROM. See ‘The Spec file' below for more details.

GV The hex byte address of the base of the global vector in the stand alone system.

HEAP The hex byte address of the heap in the stand alone system. If not specified the heap follows the global vector. The address is rounded up to a 4-byte boundary.

HEAPEND The hex byte address of the end of the heap in the stand alone system. If not specified the heap extends to the base of the display RAM. The address is rounded down to a 4-byte boundary.

REPORT When UTILCIN finishes it normally displays a report on the screen giving details of the file and stand alone environment produced. The **REPORT** argument allows this report to be written to another device, eg a disc file (as in example 2) or the printer.

The Spec file

The subsection ‘The Spec file’ in the description of ROMCIN above explains the fields in the ROM header and the format of the Spec file for ROMCIN. With the exception of the format of the ‘.ENTRY’ command that subsection applies equally well to UTILCIN and is therefore not repeated here.

Each .ENTRY command specifies one of the entry points to the ROM. A particular entry point may appear in more than one .ENTRY command. The general format of the command is:

.ENTRY name global [/A] [/M mode] [comment]

where square brackets denote optional items.

name is the '*' command for that entry point (the '*' itself need not be included);

global is the global number of the procedure to be entered (in decimal);

/A denotes that the entry point is an assembler routine ie that there is no need to initialise the BCPL interpreter before entering the code. See 'Machine code programs' in chapter 6 for more details.

/M mode specifies that a particular screen mode is to be set up before entering the program. mode must be an integer in the range 0 to 7. If '/M mode' is not specified the screen mode is left unchanged when the program is entered.

comment is any text after global not beginning **/A** or **/M.**

If a utility ROM is installed as the right-most language ROM in a BBC Microcomputer then it is entered automatically at power-on and when CTRL-BREAK is pressed. The entry point used is that specified in the first .ENTRY command in the Spec file.

Example

```
Spec file for file utilities produced by XYZ Ltd
.VERNO  3
.VERSTR 2.13
.TITLE  XYZ UTILITIES
.COPYR  (C) 1983 XYZ LTD.
.ENTRY  CATALOG 250
.ENTRY  CAT.    250             short form

.ENTRY  FORMAT  260 /A          assembler code
.ENTRY  FORM    260 /A

.ENTRY  DISPLAY 265 /M 7        uses mode 7
.ENTRY  DISP.   265 /M 7
.ENTRY  DISP80  265 /M 3        80 column version
                                   using mode 3

.ENTRY PATCH    300 /A /M 5     assembler & mode 5
```

Remarks

The file UTILINI (supplied as part of the Stand Alone Generator) is read by UTILCIN, and must therefore either be in store or on the current filing system device when UTILCIN is run.

The report produced by UTILCIN gives the following details (in both decimal words and hex bytes):

- the size of the **TO** file (ie the size of the ROM image);

- the base address and size of the global vector;

- the addresses of the heap and (if **HEAPEND** has been specified) of the end of the heap. If all the entry points to the ROM are specified in the Spec file as being assembler entry points then this information is omitted as no heap will ever be set up.

5 Procedures

This chapter shows the procedures which can be used by stand alone programs and describes any differences in the functions of the development and the stand alone versions of particular procedures.

The following table lists every procedure mentioned in chapter 5 of the User Guide for the development system. The section name is given for procedures in SYSLIB. The notes follow the table.

The section name "INOUTx" means one of the sections INOUT1 to INOUT4 - see 'I/O options' in chapter 3 for details.

PROCEDURE	SECTION	SEE NOTES
ABORT	always included	1, 2
ADVAL	ADVAL	
APTOVEC	APTOVEC	
BACKMOVE	BACKMOV	3
BACKMVBY	BACKMOV	3
CALL	always included	
CALLBYTE	always included	
CALLCO	CORTNS	
CAPCH	CAPCH	
COMPCH	COMPCH	
COMPSTRING	COMPSTR	
COWAIT	CORTNS	
CREATECO	CORTNS	
DELETECO	CORTNS	
DELFILE	DELFILE	4
DELXFILE	not available	
ENDPROG	ENDPROG	2
ENDREAD	INOUTx	5
ENDWRITE	INOUTx	5
ENVELOPE	ENVELOP	
ERRORMSG	ERRORMS	2, 6, 20

EXTSFILE	not available	
FILETOVEC	FILETOV	7
FINDARG	FINDARG	
FINDINPUT	INOUTx	4, 8
FINDOUTPUT	INOUTx	4, 8
FINDXINPUT	not available	
FINDXOUTPUT	not available	
FREEVEC	always included	
FSTYPE	INOUT4	9
GETBYTE	use OPT version	
GETVEC	always included	10
GLOBIN	GLOBIN	11
GLOBUNIN	GLOBIN	11
INPUT	INOUTx	
LEVEL	LEVEL	
LOADSEG	not available	11
LONGJUMP	always included	
MAXVEC	MAXVEC	
MODE	MODE	2, 12
MOVE	always included	
MOVEBYTE	always included	
MULDIV	always included	
NEWLINE	NEWLINE	
NEWPAGE	NEWPAGE	
OPSYS	always included	
OUTPUT	INOUTx	
PACKSTRING	use OPT version	
PUTBYTE	use OPT version	
RANDOM	RANDOM	
RDARGS	RDARGS	2, 13, 14
RDBIN	INOUTx	
RDCH	INOUTx	
RDITEM	RDITEM	13
READ	not available	
READN	READN	13
READVEC	FILETOV	7
READWORDS	READWOR	13
RENAME	RENAME	4, 15
RESUMECO	CORTNS	
RUNPROG	RUNPROG	16
SAVE	not available	
SAVEVEC	VECTOFI	17

SELECTINPUT	INOUTx	5
SELECTOUTPUT	always included	5
SHUFFLE	not available	
SOUND	SOUND	
SPLIT	SPLIT	
STACKSIZE	STACKSI	
START	user-defined	18
STARTINIT	not available	19
STOP	always included	20
TESTFLAGS	TESTFLA	
TESTSTR	TESTSTR	
TIME	TIME	
TRAP	not available	
UNLOADSEG	not available	11
UNPACKSTRING	use OPT version	
UNRDCH	INOUTx	
VDU	VDU	21
VDUINFO	VDUINFO	
VECTOFILE	VECTOFI	17
WRBIN	always included	2
WRCH	always included	2
WRITEA	not available	
WRITEBA	not available	
WRITED	WRITED	2
WRITEDB	not available	
WRITEF	WRITEF1 or WRITEF2	2, 22
WRITEHEX	WRITEHE	2
WRITEN	WRITEN	2
WRITEOCT	WRITEOC	2
WRITES	always included	2
WRITET	WRITET	2
WRITEU	WRITEU	2
WRITEWORDS	WRITEWO	13

Notes

(1) Unless the program is trapping ABORTs a message ‘ABORT’ is displayed and the program terminates. If the procedure ERRORMSG is included the message ‘Error nnn’ is also produced, where nnn is the abort code.

(2) Unless I/O options 3 or 4 have been selected any output will go to the current OSWRCH destination.

(3) If the count (converted to bytes in the case of BACKMOVE) is negative then it is treated as an unsigned value in the range 32768-65535.

(4) Filenames with no device specifier, or with a device specifier of '/F.', refer to the current filing system. A device specifier of '/S.' is invalid.

(5) Dummy procedure if I/O option 1 or 2 has been selected.

(6) In the development system ERRORMSG always produces the text 'Escape' for error 1017 instead of accessing the operating system fault text. In the stand alone system this special processing for error 1017 is not performed.

(7) If the SYSLIB2 version of this procedure is used then I/O option 4 is automatically selected (ie section INOUT4 is included). Unpredictable results will occur if NEEDS directives for other I/O options are included.

(8) The versions for I/O options 1 and 2 always return CNSLINSTR or CNSLOUTSTR.

(9) The SYSLIB1 version does not check the attributes of the current filing system, but assumes it has the attributes of a disc or Econet filing system.

(10) If there is not enough heap space available a result of 0 is given. There is no 'shuffle' facility.

(11) See 'Overlaying' in chapter 6.

(12) If the program does not need to change screen modes then the options in FIXCIN, ROMCIN and UTILCIN to set up the mode before the program is entered should be used.

(13) If this procedure is used then one of I/O options 1 to 4 should be selected.

(14) RDARGS uses the current I/O streams. Note in particular that if I/O option 2 is selected then the input to RDARGS is taken from the keyboard but is not echoed to the screen.

(15) The procedure WRITEF must be included in the program (it is not included automatically because there are two versions available).

(16) The expanded string is always passed to OSCLI, whether or not it begins with '*'. If the string contains any '%' characters (ie if it is to be expanded) then an appropriate version of WRITEF must be included in the program. If the expanded string is longer then 255 characters it is truncated. If the string contains a '*N' character then it is treated simply as line feed, not as line feed/carriage return.

(17) The second parameter must be a filing system file. Thus:

```
SAVEVEC( myvec, "/P")
VECTOFILE( myvec, "/L")
```

are not allowed.

(18) START must be included in a stand alone program created by FIXCIN or ROMCIN. It need not be included in one created by UTILCIN.

(19) If STARTINIT is included in a stand alone program it is ignored.

(20) Error and warning messages are only produced if ERRORMSG is included.

(21) Outputs to the current OSWRCH device.

(22) WRITEF1 supports %I, %N and %S only. WRITEF2 supports %C, %I, %N, %S, %X and %$.

6 Discussion

This chapter discusses various topics concerned with the stand alone environment and with specific problems that may be encountered when creating stand alone programs.

The following subjects are covered:

ESCAPE handling

Handling '*' commands

Location of global vector and heap

Machine code programs

Overlaying

Program termination

Programs to run on non-BBC machines

Replacing SYSLIB procedures

Writing programs to execute in ROM

ESCAPE HANDLING

All stand alone programs are entered with the ESCAPE key generating an ASCII code instead of causing an ESCAPE condition. This has some advantages (in particular it is not necessary to test for an ESCAPE condition whenever OSRDCH or OSWORD(0) is called), but also has disadvantages - in particular:

(1) there is no simple way for the user to interrupt a stand alone program;

(2) there is no way for the user to abandon slow I/O operations (eg there is no way to terminate '*CAT' on the tape filing system, or to persuade the system to give up the search for a particular tape file).

A program can re-enable the normal ESCAPE effects and subsequently disable them again by

OPSYS(229, x, 0)

where x is 0 to re-enable ESCAPE and 1 to disable it. If a program does enable ESCAPE, however, then it must itself handle the effects of ESCAPE being pressed (in the development system these effects are handled by the system procedures).

If ESCAPE is pressed when the normal ESCAPE effects are enabled then an escape condition is declared. This sets the top bit in byte 255. In addition:

(1) if the computer is currently executing an OSRDCH call or an OSWORD call to read a line of input then the C bit is set and the character returned is 1B hex. The library procedures in SYSLIB do not check for this.

(2) if the computer is currently executing certain other operating system I/O routines (eg to open a cassette tape file) a fault condition is declared and the BCPL fault routine is entered. The fault routine acknowledges the escape condition and sets MCRESULT to #XFF11. The subsequent action is up to the routine that called the operating system routine. Library routines that can return a failure indication to the caller (eg FINDINPUT) do so. Those that cannot (eg RDCH) call ABORT. In both cases the error number (in RESULT2) or the abort code is 1017.

(3) until the escape condition is acknowledged any subsequent calls to the routines mentioned in (1) and (2) above have the effects described there.

The processing required in the program to handle these effects is as follows:

(1) If ESCAPE is to be used as a means of interrupting the program then the program should periodically check for an escape condition and, if it exists, acknowledge it and take appropriate action. Suitable code might be:

```
IF 0%255 > 127 THEN  // escape condition
$( OPSYS(126)        // acknowledge it
   ...               // take action
$)
```

(2) After each read from devices /C, /K or /P the program should check for an escape condition. If an escape condition is present it should be acknowledged and suitable action taken. This might include repeating the read. The check must also be done after calling library procedures which read from these devices, eg READN, RDITEM, RDARGS and ENDPROG.

(3) After each call to a library procedure which uses operating system I/O routines and which returns an error indication, the program should check specially for error 1017 and take the appropriate action.

(4) ABORTs should be trapped. If ABORTCODE is 1017 appropriate action should be taken.

The details of the action to be taken by the program when it detects that ESCAPE has been pressed are obviously specific to the program.

HANDLING '*' COMMANDS

In the development system the existence of the command state allows the user to enter various '*' commands, eg to display a disc directory or to set up the baud rate of the serial output. He can even interrupt running programs to do this.

In the stand alone environment there is no command state and so stand alone programs may have to make explicit provision for the user to enter '*' commands. This is less important with programs created by FIXCIN, since the user can presumably enter any necessary '*' commands before entering the '*RUN' command to run the program, but is very important with programs created by ROMCIN or UTILCIN.

Such programs may be installed as the right-most language ROM and are therefore entered when the computer is switched on. Unless they allow '*' commands to be entered then the user will never be able to access any other language ROMs in the machine.

There are two approaches that may be adopted:

- Prompt the user for specific options and call OPSYS or OSCLI from within the program. For example, a program about to use the printer might prompt for the printer ignore character, serial or parallel printer and (if serial) the baud rate.

- Allow the user to enter any '*' command and pass the command directly to OSCLI. This method is particularly convenient if the program is one that periodically gets input from the user; the recommended convention is that if the input begins with '*' then the whole line is given to OSCLI.

If a program adopts the second method above, and is reading the user input from device /C, then it is not necessary to copy the input line into a special buffer before calling OSCLI. Instead the address of the buffer used by the BCPL I/O can be passed to OSCLI as in the following example:

```
GLOBAL   $( CISBUF:49 $)
MANIFEST $( OSCLI=#XFFF7 $)

IF /* first character of line is '*' */ THEN
$( RESULT := CALLBYTE(OSCLI, 0, CISBUF << 1)

   etc.
$)
```

LOCATION OF GLOBAL VECTOR AND HEAP

When creating a stand alone program the location of the global vector in the target system must always be specified. The location of the heap may optionally be specified as well.

If the location of the heap is not specified then it follows the global vector (for programs in language ROMs) or follows the program, which itself follows the global vector (for programs created by FIXCIN). In the latter case the heap overwrites the initialisation code and data. In both cases the heap extends to the bottom of the display RAM. (See the Appendix for details of the heap when a program is running in a 6502 second processor.)

To make best use of this default arrangement the global vector should be located at as low an address as possible, subject to its not overwriting the operating system/filing systems workspace. Any RAM between the top of this workspace and the base of the global vector is not used by the BCPL system (although individual programs can access it directly if required).

When generating a stand alone program which is intended to be run only on machines with a particular configuration of filing systems, then the highest workspace address is known and the global vector can be located immediately above it (eg for a machine with no filing system other than cassette the global vector can be located at hex byte address 0E00; for a machine with the standard disc filing system the corresponding address is hex byte 1900).

When generating a stand alone program to be run on a variety of machines with a variety of different filing systems then a relatively high address must be specified for the global vector, wasting space on those machines with fewer filing systems. To help overcome this problem a procedure is provided (in source form) which links the area from the top of the workspace to the bottom of the global vector into the heap.

This procedure is supplied in the file EXTHEAP. The code is commented and should be self-explanatory. It allows a number of words to be reserved above the workspace eg to allow for user-defined characters.

If a program uses EXTHEAP then it should call it at an early stage of processing. It is quite a good idea to position the global vector fairly close to the bottom of the display RAM (though this involves knowing whether the program is to be run on a Model A or a Model B version of the BBC Microcomputer) so that the initial heap is quite small. Then when EXTHEAP is called a large contiguous area will be obtained.

Note that the initial heap must be big enough for any I/O streams set up during system initialisation (with I/O options 1, 2, 3 and 4 approximately 100 words, 30 words, 120 words and 120 words respectively are needed).

Various other arrangements of the global vector, the heap and the program are possible. The description of the heap format given in the Appendix provides the necessary information for writing procedures such as EXTHEAP.

MACHINE CODE PROGRAMS

The mechanism of calling machine code procedures from stand alone BCPL programs is identical to the mechanism used in the development system. This section is mainly concerned with programs written entirely in machine code. It also covers the case of machine code entry points in utility ROMs which may also contain CINTCODE entry points.

Starting

The initialisation performed when a stand alone machine code program is started is similar to that performed for a BCPL program as described in chapter 3, with the following differences:

- no stack is set up;

- no heap is set up;

- the interpreter is not initialised;

- if the global procedure FAULTROUTINE is included in the program (or the utility ROM) then this is set up as the fault routine (ie the byte address of the routine is set up in $202/$203).

The machine code program is entered at START (or, in the case of programs generated by UTILCIN, at the defined global procedure). If the first byte of the procedure is $0D or $CF (the two special bytes used for starting machine code procedures to be called from BCPL) then the procedure is entered at the second byte.

If a fault routine has not been set up automatically then the program should set one up before making any operating system calls that might generate faults.

Stopping

As a general rule a machine code program must handle termination itself. It cannot, for example, exit by any of the JMPs to page 4 locations that are available to machine code routines called by CINTCODE and it certainly must not simply RTS (this will give unpredictable results).

It is possible, however, to include one of the termination options described in the section 'Program termination' below and to jump to it by code such as:

```
ENDMSG  GLOBAL  149          ; global number of
                             ;  term. option
WORK    EQU     $400         ; work area

        LDA     ENDMSG       ; convert word address
        ASL     A            ;  to byte address in
        STA     WORK         ;  work area
        LDA     ENDMSG+1
        ROL     A
        STA     WORK+1
        LDA     #0           ; parameter for term.
        PHA                  ;  option
        JMP     (WORK)       ; enter term. option
```

The advice given on tidying up in 'Program termination' below applies to machine code programs (in particular ESCAPE should be re-enabled).

Use of RAM

A stand alone machine code program may use all the RAM areas normally reserved for the current language - in particular bytes $00 to $8F in the zero page and all of pages 4 to 7. Additionally all the RAM from the top of the operating system workspace to the bottom of the display RAM is available, apart from the area used for the global vector and (for programs generated by FIXCIN) the area in which the program itself resides.

A machine code program can get the address of the global vector by code such as:

```
MAXGLOB GLOBAL  0
GVADDR: DW      MAXGLOB

        LDX     GVADDR
        LDY     GVADDR+1
```

which sets up X/Y with the byte address of the global vector. The length of the global vector can be calculated from MAXGLOB.

A machine code program created by FIXCIN can find its start and end addresses by code similar to the following:

```
STRLAB:          ; MUST be first line of code

CODSTR: DW       STRLAB
CODEND: DW       ENDLAB

        LDX      CODSTR  ; X/Y contain start addr
        LDY      CODSTR+1

ENDLAB:          ; MUST be last line of code
```

Of course, if the machine code program is in several sections then care must be taken to JOINCIN them in the right order (and if NEEDCIN is used to extract sections from a library then the section containing ENDLAB must be JOINCINed onto the end of the output from NEEDCIN). It will also be necessary to declare STRLAB and ENDLAB as globals.

Generating machine code programs

The stages in generating a stand alone machine code program are similar to the stages for generating a stand alone BCPL program, described in chapter 2. The main difference is that no sections from SYSLIB will be included (except perhaps for a termination option).

When using FIXCIN or ROMCIN the **NOINT** argument must be specified. When using UTILCIN the **/A** parameter must be specified on each '.ENTRY' statement in the Spec file.

OVERLAYING

The development environment provides the procedures LOADSEG, GLOBIN, GLOBUNIN and UNLOADSEG which provide a simple method of reading overlays into store, linking them and unlinking them.

LOADSEG and UNLOADSEG are not provided in the stand alone environment, but overlays can still be handled by reading CINTCODE files into vectors (using READVEC or FILETOVEC). Both GLOBIN and GLOBUNIN take the address of the vector as a parameter. Thus code to use an overlay in the CINTCODE file "OVER1" (on the current filing system) might be:

```
LET OVPTR = FILETOVEC("OVER1")
TEST OVPTR NE 0 &       // read OK &
     GLOBIN(OVPTR) THEN // linked OK
$(
   code to use the overlay

   GLOBUNIN(OVPTR)      // unlink it
   FREEVEC(OVPTR)       // and return the vector
$)
```

Note that FILETOVEC and READVEC using SYSLIB2 force I/O option 4 to be used.

The principal differences in the use of GLOBIN and GLOBUNIN, compared with the development system are:

- there is no way to relink the procedures in the program. Thus if an overlay needs to temporarily redefine a procedure in the program it must save and restore the old value. For example

```
LET OLDRDCH = RDCH
RDCH := LOCAL.RDCH

code using the redefined RDCH

RDCH := OLDRDCH
```

- there is no chaining of linked files. Thus GLOBUNIN always returns a result of 0;

- it is not necessary to use GLOBUNIN before releasing or re-using a vector containing an overlay, provided that it is guaranteed that none of the procedures in that overlay will be called again. One advantage of using GLOBUNIN, however, is that a program can compare a global with GLOBWORD to decide if an overlay should be loaded.

If a program uses several overlays in turn it is recommended that it allocates a vector for use as an 'overlay bay'. The vector should be large enough to hold the biggest overlay used. This strategy ensures that overlays can always be loaded, regardless of other demands for heap space.

The size of the global vector created in a stand alone environment is governed by the highest global number used or declared in the program. If an overlay is to use higher global numbers than those used in the program then the program must contain a reference to the highest global required.

It is not sufficient to just include a statement such as

```
GLOBAL $( HIGLOB:532 $)
```

The global must be referred to in the code, eg as a label.

Care must be taken when creating overlays containing library procedures from SYSLIB. In particular an overlay should not contain a procedure which is also contained in the root. (Once the overlay has been linked the global vector entry will point to the procedure in the overlay rather than to the one in the root. When the overlay is unlinked or overwritten the procedure can no longer be called by the root.)

Note that some sections in SYSLIB contain NEEDS directives for other sections. If it is required to include a section in an overlay without including the sections NEEDed by it, the overlay should contain dummy versions of those sections.

‘Sections in SYSLIB’ in the Appendix lists the sections NEEDed by other sections.

For example, an overlay might require ENDPROG. The section ENDPROG includes a NEEDS directive for CAPCH, which might be required in the root. The code for the overlay might start:

```
SECTION "CAPCH"
// dummy section
.
SECTION "OVERLAY"
NEEDS "ENDPROG"
```

PROGRAM TERMINATION

Overview of program termination

When a BCPL stand alone program terminates by calling STOP, calling ABORT, calling ENDPROG, executing FINISH or returning from START (or the entry point procedure in the case of utility ROMs) the system procedure ENDINT (global 8) is called.

If a stand alone program terminates with a fatal interpreter error (eg run out of stack, call to uninitialised global) then ENDINT is entered from the interpreter.

ENDINT is a machine code procedure which re-enables ESCAPE then checks to see if a termination option has been included in the program. If so the termination option is entered. If not then ENDINT just loops - the only way out is BREAK.

Three termination options (described below) are provided in SYSLIB. It is also possible to include a user-written termination option. As a general rule any program which exits by calling STOP etc should include a termination option. If the program always exits by some other method (eg by prompting the user for a '*' command), or never exits then there is no need for a termination option, other than to catch fatal interpreter errors. The latter should not occur in fully debugged programs.

Standard termination options

A termination option is included in a program by extracting the appropriate section from SYSLIB with a NEEDS directive.

DIAGMSG

This option is useful in the early stages of testing a stand alone program since it provides diagnostic information if there is a fatal interpreter error.

OSRDCH and OSWRCH are reset to read from the keyboard and write to the screen. If a fatal interpreter error has occurred then

FATAL ERROR x

is displayed, where x is the error letter, followed by a display (in hex) of the CINTCODE P, PC and A registers and the 8 CINTCODE instructions immediately preceding the address in the PC. Note that the P register is the word address of the current stack frame whereas the PC register is the byte address of the next CINTCODE instruction to execute.

In many cases the information displayed will be sufficient to locate the cause of the error. In particular if the error is error G (calling an uninitialised global) then disassembling the CINTCODE will usually reveal which procedure is being called.

Whether or not a fatal interpreter error has occurred 'PROGRAM ENDED - ENTER '*' COMMAND' is displayed followed by a prompt ('*'). When the user has entered a line of input it is passed straight to OSCLI. If the line is a command to enter a language ROM (eg 'BASIC') then that ROM is entered and the language started up. If not then the prompt ('*') is re-issued and the process repeated.

ENDMSG

This option is identical to DIAGMSG except that if a fatal interpreter error has occurred the CINTCODE P, PC etc are not displayed.

It is probably the best of the three options for general use.

ENDRST

This option re-enters the current language, unless the program is running in the 6502 second processor when it sets OSWRCH to output to the screen, displays 'Please press BREAK' and sits in a loop.

It is likely to be of most use in stand alone programs generated by FIXCIN which are not expected to be run in a second processor, since it will restart the language from which the '*RUN' command was issued. In the case of programs generated by ROMCIN and UTILCIN re-entering the current language simply restarts the program.

Writing termination options

User-written termination options may be included simply by writing appropriate assembler code, assembling it using RAS and including the resultant code in the program.

The assembler code must define global 149. ENDINT enters the termination option by JMPing to this global. The top byte of the stack contains 0 if the program has ended by calling STOP etc or the error letter (ASCII) if it has ended by calling a fatal interpreter error. A typical termination option might start like this:

```
TERMIO  GLOBAL  149

TERMIO: PLA             ; normal exit ?
        BEQ     ISNORM  ; branch if yes
        CMP     #'G'    ; uninitialised global ?
etc.
```

On entry to the termination routine the OSRDCH source and OSWRCH destination may not be the defaults (keyboard and screen). ESCAPE is enabled. The fault routine is the BCPL fault routine. The termination routine should replace this by its own fault routine before doing anything that might cause a fault. (The byte address of the fault routine is stored in bytes 202/203 hex.)

Tidying up

When a BCPL program running in the development system terminates the development environment automatically tidies up after it. In the stand alone environment there is no automatic tidying up.

Before exiting a program should perform as many of the following tidying up operations as are appropriate (if the program exits via ENDINT then ESCAPE will be re-enabled and some of the termination options also reset the OSRDCH and OSWRCH assignments):

- re-enable ESCAPE;
- reset OSRDCH to read from the keyboard;
- reset OSWRCH to write to the screen;
- close all open files;
- re-enable cursor editing;
- reset function keys to generate strings;
- turn off the printer.

PROGRAMS TO RUN ON NON-BBC MACHINES

FIXCIN can be used to generate BCPL or assembler programs which can be executed on target machines other than the BBC Microcomputer. The output from FIXCIN is a file containing code designed to be loaded and executed at a certain address in the target machine's address space. If appropriate the code can be held and executed in ROM.

Target machine requirements

The target machine must satisfy the following requirements:

- the processor must be a 6502;

- the following hex byte addresses must exist in RAM:

  0000 to 0059
  0100 to 01FF
  0400 to 07FF (only for BCPL programs)

  Additionally the areas for the global vector and (for BCPL programs only) the heap must exist in RAM;

- there must be a way of entering the program (at the address specified by the **BASE** parameter to FIXCIN).

Restrictions on the program

The program is entered in the same way as normal, except that the initialisation code does not make the operating system call to disable ESCAPE. The way the program exits must be specific to the application; the standard termination options in SYSLIB must not be used since they make operating system calls.

The program must not use BCPL I/O (as the I/O routines use operating system calls). In particular none of the four I/O options must be included (since they all cause operating system calls to be made during program initialisation).

The program must not call any operating system routines and must not call any library procedures that call operating system routines.

If the program is to be put into ROM it must conform to the restrictions detailed in 'Writing programs to execute in ROM' later in this chapter.

FIXCIN arguments

The argument **NOTBBC** must be specified. Additionally **BASE, HEAP** and **HEAPEND** must all be specified. **MODE** may not be specified.

REPLACING SYSLIB PROCEDURES

The normal method of generating a stand alone program is to JOINCIN the various sections of the program into one file then use NEEDCIN to append the required SYSLIB sections. Thus the SYSLIB sections come after the program sections and so if the same global is defined in the program and in the included SYSLIB sections the SYSLIB definition will be the one actually used.

If the program is to redefine SYSLIB procedures such as ABORT (always included) or FSTYPE (always included if I/O option 4 is selected) the section(s) containing the redefined procedures must be JOINCINed with the rest of the program after NEEDCIN has been used to extract the SYSLIB sections.

WRITING PROGRAMS TO EXECUTE IN ROM

This section lists some restrictions that must be observed when writing programs which are to execute in a language ROM.

- STATIC data, TABLEs and strings cannot be modified (since they are all held with the CINTCODE). Thus there is little point in using STATIC data at all.

The restriction on modifying strings means that

```
LET X = "CAR"
X%3 := 'T'
```

will not work, but

```
LET X = "CAR"
X := "CAT"
```

will work.

- Assembler routines must not include any data that is to be changed with the code. Thus

```
COUNT:  DB      0
        INC     COUNT
```

will not work.

- The method for calling global procedures from machine code recommended in the development system User Guide cannot be used. One possible method of calling the global procedure PROCB is

```
LDA     #$4C  ; JMP instruction
STA     TEMP
LDA     PROCB ; convert to byte addr
ASL     A
STA     TEMP+1
LDA     PROCB+1
ROL     A
STA     TEMP+2
JSR     TEMP  ; JSR to JMP instruction
```

where TEMP is a suitable 3-byte area.

- Any strings or parameter blocks which are to be passed to operating system routines must be in RAM and not in the ROM. The reason is that other sideways ROMs may be invoked to deal with the call and they will interpret the address of the string or block as being an address in their own ROM.

  Thus an attempt to restart BASIC using OSCLI such as

```
MANIFEST $( OSCLI = #XFFF7 $)
LET STRING = "**BASIC*C"
CALLBYTE(OSCLI, ?, (STRING << 1) + 1)
```

  will fail. Instead code such as

```
MANIFEST $( OSCLI = #XFFF7 $)
LET STRING = VEC 3
MOVE("**BASIC*C", STRING, 4)
CALLBYTE(OSCLI, ?, (STRING << 1) + 1)
```

  should be used.

  All library routines such as FINDINPUT, DELFILE, RUNPROG etc copy their parameters into RAM, so it is safe to code

```
RUNPROG("**BASIC")
```

  (The final '*C' is not needed as RUNPROG adds it automatically.)

7 Summaries

This chapter summarises the features of the BCPL Stand Alone Generator under the following headings:

Error numbers and fatal error codes

Files provided

Global variables

Global vector

Utilities

Utility error messages

ERROR NUMBERS AND FATAL ERROR CODES

This section lists all the error numbers and codes that can be generated in the stand alone environment. A comparison of this list with the corresponding list in the development system User Guide will show which error checks have been omitted in the stand alone environment.

Error numbers

15 MODE - cannot free necessary heap space to change mode.

18 RENAME - illegal device specified (eg /L).

23 DELETE - illegal device specified (eg /P).

27 file does not exist in current filing system.

50 READVEC, FILETOVEC - 'from' device is not the current filing system;
SAVEVEC, VECTOFILE - 'to' device is not the current filing system.

51 not enough store to get vector or vector too small. This error is generated by GETVEC if it fails to get a vector and also by various other library procedures if they cannot get the vector(s) they need;
READVEC - the supplied vector is not big enough.

52 device not recognised. A device/file name begins with '/' but the next character is not one of the devices recognised by the system. This error is also generated if the '.' separating ''/F' from the filename is omitted.

55 GLOBIN/GLOBUNIN - vector does not contain a valid CINTCODE file.

56 GLOBIN - unable to link. The vector contains a hunk which uses or defines a global variable beyond the end of the global vector.

60 FINDINPUT/FINDOUTPUT - attempt to use input device for output or vice versa.

101 cannot output.

122 GETVEC - heap corrupt. The program is unlikely to be able to recover from this error.

123 APTOVEC - no room in stack.

Error numbers in the range 1000-1255 correspond to the operating system error numbers 0-255. See the BBC Microcomputer User Guide and the relevant filing system User Guides for full details of these errors.

Fatal error codes

If the interpreter detects an error it enters the termination option with the error letter on the 6502 stack (see 'Program termination' in chapter 6). The error letters are:

G Call to uninitialised global.
S Stack overflow.
X Invalid CINTCODE instruction.
Z Attempt to divide by zero.

FILES PROVIDED

This section lists the files provided with the Stand Alone Generator.

B.EXTHEAP the source of EXTHEAP (procedure to extend the heap in a stand alone environment).

FILETRN program to copy a file preserving the load and execution addresses.

FIXCIN program to create a stand alone program as a file run by '*RUN'.

FIXINI initialisation code read by FIXCIN.

PACKCIN program to compact CINTCODE files.

ROMCIN program to create a stand alone program as a language ROM.

ROMINI initialisation code read by ROMCIN.

SYSLIB1 procedure library (assuming disc- or Econet-like filing system).

SYSLIB2 procedure library (supporting all filing systems).

UTILCIN program to create a stand alone program as a utility ROM.

UTILINI initialisation code read by UTILCIN.

GLOBAL VARIABLES

This section lists the global variables which may be useful in stand alone programs. The global number and 'L' for LIBHDR or 'S' for SYSHDR are given in brackets after the global name. Unless otherwise stated these variables are used in exactly the same way as they are used in the development system.

ABORTCODE (4 S)
ABORTLABEL (5 S)
ABORTLEVEL (6 S)

CNSLINSTR (52 L)
    Provided one of the four I/O options is included this is initialised by the system to a console input stream. It may be selected by a program whenever input from the console is required. Note that with I/O option 2 the console is read as device /K (ie a key at a time with no echo) whereas with options 1, 3 and 4 it is read as device /C (a line at a time with echo). Set to GLOBWORD if no I/O option is selected.

CNSLOUTSTR (53 L)
    Provided one of the four I/O options is included this is initialised by the system to a console output stream. It may be selected by a program whenever output to the screen is required. Set to GLOBWORD if no I/O option is selected.

CURRCO (22 S)

ERRORSTREAM (58 S)
    Provided that I/O option 3 or option 4 is included then this is set up as a dummy stream. It is the stream that is selected for input following ENDREAD and for output following ENDWRITE. Also selected by SELECTINPUT(0) and SELECTOUTPUT(0). An attempt to read from or write to this stream causes unpredictable results. If neither I/O option 3 nor option 4 is included then ERRORSTREAM is set to GLOBWORD.

LIBBASE (117 S)
    Initialised to 0. Included for compatibility with the development system.

MAXGLOB (0 S)

MCRESULT (11 L)

RESULT2 (15 L)

SYSINDEX (18 S)
    Pointer to an 8-word table initialised to zeros. Included for compatibility with the development system.

GLOBAL VECTOR

The following globals declared in LIBHDR and SYSHDR must not be used by stand alone programs:

CLIINSTR	CONTPRG	DELXFILE
ENDTRAP	EXTSFILE	FINDXINPUT
FINDXOUTPUT	LASTERROR	LINKEDFILES
LOADSEG	MAINSTACK	READ
SAVE	SHUFFLE	STARTINIT
STOREFILES	TIDYSTATE	TRAP
TRAPSTACK	TRAPSTART	UNLOADSEG
WRITEA	WRITEBA	WRITEDB

Dummy versions of the globals LIBBASE and SYSINDEX are provided.

The first free global for general use is number 250.

UTILITIES

FILETRN - file transfer
    FROM/A,TO/A,FROMFS,TOFS,PAUSE/S

FIXCIN - create program to run in RAM
    FROM/A,TO/A,GV/A/K,BASE/K,HEAP/K,HEAPEND/K,
    MODE/K,MAX/S,NOTBBC/S,REPORT/K,NOINT/S

PACKCIN - pack CINTCODE
    FROM/A,TO/A,REPORT/K

ROMCIN - create program as language ROM
    FROM/A,TO/A,SPEC/A,GV/A/K,HEAP/K,HEAPEND/K,
    MODE/K,REPORT/K,NOINT/S

Spec file commands are:

```
.COPYR  <string beginning '(C)'>
.ENTRY  <name> [comments]
.TITLE  <string>
.VERNO  <number> [comments]
.VERSTR <string>
```

UTILCIN - create program as utility ROM
FROM/A,TO/A,SPEC/A,GV/A/K,HEAP/K,HEAPEND/K,
REPORT/K

Spec file commands are:

```
.COPYR  <string beginning '(C)'>
.ENTRY  <name> <global> [/A] [/M <mode>]
        [comments]
.TITLE  <string>
.VERNO  <number> [comments]
.VERSTR <string>
```

UTILITY ERROR MESSAGES

The majority of the error messages produced by the utilities are self-explanatory. Those that are not are listed in alphabetical order below.

CANNOT FIND 'FROM' FILE LENGTH
The filing system from which the file is being copied does not support the OSFILE call to find a file's length.

CANNOT SAVE xxx
Unless the **MAX** argument is specified the stand alone program file is created in store then written to the current filing system using the procedure SAVE. This message indicates that the SAVE has failed (**xxx** is the file name).

CANNOT SELECT 'FROM' FILING SYSTEM
An error has occurred selecting **FROMFS**. The most likely reason is that **FROMFS** is not a valid '*' command.

CANNOT SELECT 'TO' FILING SYSTEM

An error has occurred selecting **TOFS.** The most likely reason is that **TOFS** is not a valid '*' command.

'FROM' FILE IS STORE FILE

A file named **FROM** exists in store and cannot be deleted.

GLOBAL xxx ENTRY POINT OMITTED

A '.ENTRY' command in the Spec file specifies an entry point as global number **xxx**, but that global is not defined in the **FROM** file.

HEAP SIZE 0 OR -VE

The address specified for **HEAPEND** is not at least four bytes above that specified (or calculated) for the heap.

INVALID '.COPYR' ON LINE xxx OF SPEC FILE

The parameter for '.COPYR' does not begin with '(C)'.

INVALID NUMBER ON LINE xxx OF SPEC FILE

Either a number has been omitted where one is expected (eg in a command such as

```
.ENTRY FRED /A
```

where the global number is missing) or a number has been specified but is out of range eg an invalid screen mode such as

```
.ENTRY FRED 250 /M 8
```

INVALID RELOCATION DATA IN xxx

The file **xxx** (normally the **FROM** file) contains assembler code which is to be relocated relative to the globals COMMON2 or COMMON3.

NEED 'BASE', 'HEAP' & 'HEAPEND' WITH 'NOTBBC'

The argument **NOTBBC** has been specified without specifying all of **BASE**, **HEAP** and **HEAPEND.**

PROGRAM TOO BIG

The ROM image being created is larger than 8192 words, which is the maximum size for a language ROM.

'SAINIT' UNDEFINED

The initialisation file FIXINI, ROMINI or UTILINI does not include the system procedure SAINIT. The most likely cause is that a private version of the initialisation file has been used.

SECTION 'INTERP' INCLUDED BUT NO BCPL ENTRY POINTS

Despite the **/A** parameter being specified for all the '.ENTRY' commands in the Spec file, indicating that the program is written entirely in assembler, the BCPL interpreter has been included in the **FROM** file.

SECTION 'INTERP' INCLUDED BUT 'NOINT' SPECIFIED

Despite the **NOINT** argument having been specified, indicating that the program is written entirely in assembler, the BCPL interpreter has been included in the **FROM** file.

SECTION 'INTERP' MISSING

The **FROM** file does not include the BCPL interpreter. Either the **NOINT** argument should have been specified (for UTILCIN the **/A** parameter should have been specified for all the '.ENTRY' commands in the Spec file) or the NEEDS "INTERP" directive has been omitted from the program (or the stage of extracting sections from SYSLIB has been omitted).

'START' UNDEFINED

The **FROM** file does not include the global procedure START.

‘SYSINIT’ UNDEFINED

This error indicates that the BCPL interpreter is included in the **FROM** file but that the section "RTPROCS" has been omitted. Since the section "INTERP" includes a NEEDS directive for "RTPROCS" the most likely cause is that a private version of the interpreter or of SYSLIB has been used.

UNRECOGNISED COMMAND ON LINE xxx OF SPEC FILE

A line in the Spec file begins with ‘.’ but is not one of the five valid commands.

Appendix

This Appendix contains a number of sections describing various aspects of the use of the Stand Alone Generator and of the stand alone environment which may be of interest to the more advanced user.

It includes details of the stand alone environment in a 6502 second processor, notes on how to estimate the size of a stand alone program in advance (though it may well be easier just to create the program and see how big it is), descriptions of the output files from FIXCIN, ROMCIN and UTILCIN and specifications of the system data areas and procedures used in the stand alone environment.

This Appendix is divided into the following sections:

Globals used by Stand Alone Generator

Re-use of global numbers

Running in the 6502 second processor

Sections in SYSLIB

Sizes

Stand alone file formats

System data areas

System procedures

GLOBALS USED BY STAND ALONE GENERATOR

The three programs FIXCIN, ROMCIN and UTILCIN make use of three special globals when generating stand alone programs.

SAINIT (206)
: This global should be defined in the initialisation file (FIXINI, ROMINI or UTILINI). It is the assembly language procedure that is JMPed to when the stand alone program is executed and which sets up the global vector, stack, heap etc.

  The global is only used during the process of generating the stand alone program and is not set up in the global vector in the stand alone environment.

DSPTAB (207)
: This global is defined in section "INTERP" in SYSLIB. It marks the start of the jump tables used by the despatch routine in the interpreter.

INTERP (208)
: This global is defined in section "INTERP" in SYSLIB. It is the entry point to the BCPL interpreter. FIXCIN, ROMCIN and UTILCIN use this global to decide whether or not the interpreter is included in the **FROM** file.

RE-USE OF GLOBAL NUMBERS

Some of the global numbers allocated to system data areas and procedures in the development system have been re-used for different data areas and procedures in the stand alone environment. It is thus very important that stand alone programs do not attempt to use these globals with their old meanings.

The global numbers affected are as follows. The majority of them are described in the Appendix in the development system User Guide, and the names used for them there are also listed.

```
  7  PRGENDLEVEL
  8  PRGENDLABEL
 16  STOREFILES
102  EXTSFILE (library procedure in development
     system)
113  SFCNTRL
119  -
149  CONTPRG
206  RUNSUSP
207  EXERROR
208  TIDYSTATE
```

RUNNING IN THE 6502 SECOND PROCESSOR

A stand alone BCPL program can normally run with no alterations in the 6502 second processor. If the program was created by FIXCIN then the file is automatically loaded into the second processor by the '*RUN' command. If the program is in a language ROM then the code in the ROM is automatically copied over to the second processor by the operating system.

The main effects of running in the second processor are that more RAM is available and that processing speed is significantly increased. Provided that the **HEAPEND** argument was not specified when the program was created the initialisation code uses this extra RAM to extend the heap as follows:

- for programs created by FIXCIN the heap extends from the top of the program (or from the address specified by the **HEAP** argument) up to hex byte address F000. Note that the heap therefore overwrites the copy of the current language ROM so it is not possible to re-enter the current language by pressing BREAK or by:

  JMP $8000

- for programs created by ROMCIN/UTILCIN the heap extends from the top of the global vector (or from the address specified by the **HEAP** argument) up to hex byte address F000, but with the area occupied by the program (ie hex byte address 8000 up to some address less than or equal to C000) marked as an allocated vector.

The only other feature of the system that appears different is that the heap is not adjusted by the MODE procedure and so error 15 (not enough heap space to change display mode) can never occur.

SECTIONS IN SYSLIB

The following table contains one entry for each section in SYSLIB, showing the procedures contained in that section, the other SYSLIB sections NEEDed by that section and the approximate size of the code plus global relocation data in that section (decimal words).

SECTION	PROCEDURES	NEEDS	SIZE
ADVAL	ADVAL	-	9
APTOVEC	APTOVEC	-	30
BACKMOV	BACKMOVE, BACKMVBY	-	43

SECTION	PROCEDURES	NEEDS	SIZE
CALLOSF	System procedure used by READVEC/FILETOVEC/ SAVEVEC/VECTOFILE	-	75
CAPCH	CAPCH	-	10
CHANGEC	CHANGECO	-	64
COMPCH	COMPCH	CAPCH	7
COMPSTR	COMPSTRING	COMPCH	33
CORTNS	CALLCO, COWAIT, CREATECO, DELETECO, RESUMECO	CHANGEC	79
DELFILE	DELFILE	FNAME	48
DIAGMSG	Termination option	-	137
ENDMSG	Termination option	-	74
ENDPROG	ENDPROG	CAPCH	34
ENDRST	Termination option	-	40
ENVELOP	ENVELOPE	-	19
ERRORMS	ERRORMSG	WRITEN	49
FILETOV (SYSLIB1)	FILETOVEC, READVEC	CALLOSF FNAME OPENMSG	67
FILETOV (SYSLIB2)	FILETOVEC, READVEC	CALLOSF INOUT4 MAXVEC	144
FINDARG	FINDARG	COMPCH	50
FNAME	FILENAME, NAMESTR	FINDARG	75

GLOBIN	GLOBIN, GLOBUNIN	-	178
INOUT1	Various I/O procedures	IOCOM	76
INOUT2	Various I/O procedures	IOCOM	65
INOUT3	STRCNTL	IODEV FINDARG	160
INOUT4 (SYSLIB1)	STRCNTL	FNAME IODEV OPENMSG	244
INOUT4 (SYSLIB2)	STRCNTL	FNAME IODEV OPENMSG	294
INTERP	BCPL interpreter, CALL, CALLBYTE, MOVE, MOVEBYTE, MULDIV, OPSYS	RTPROCS	1346
IOCOM	INPUT, OUTPUT, RDBIN, RDCH, UNRDCH	-	68
IODEV	Various I/O procedures for I/O options 3 & 4	IOCOM	215
LEVEL	LEVEL	-	5
MAXVEC	MAXVEC	-	23
MODE	MODE	-	49
NEWLINE	NEWLINE	-	4
NEWPAGE	NEWPAGE	-	5
OPENMSG (SYSLIB1)	FSTYPE	-	11
OPENMSG (SYSLIB2)	FSTYPE	-	89

RANDOM	RANDOM	-	6
RDARGS	RDARGS	COMPSTR FINDARG RDITEM	169
RDITEM	RDITEM	CAPCH	103
READN	READN	-	41
READWOR	READWORDS	WDIOCOM	79
RENAME	RENAME	FNAME RUNPROG	63
RTPROCS	ABORT, FREEVEC, GETVEC, LONGJUMP, STOP, SYSINIT	-	116
RUNPROG	RUNPROG	-	88
SOUND	SOUND	-	7
SPLIT	SPLIT	-	34
STACKSI	STACKSI	-	8
TESTFLA	TESTFLAGS	-	20
TESTSTR	TESTSTR	-	14
TIME	TIME	-	12
VDU	VDU	-	122
VDUINFO	VDUINFO	-	28
VECTOFI	SAVEVEC, VECTOFILE	CALLOSF OPENMSG FNAME	44
WDIOCOM	Common routine for READWORDS & WRITEWORDS	-	71

WRITED	WRITED	-	60
WRITEF1	WRITEF	CAPCH WRITED	74
WRITEF2	WRITEF	CAPCH WRITED WRITEHE	91
WRITEHE	WRITEHEX	-	25
WRITEN	WRITEN	WRITED	5
WRITEOC	WRITEOCT	-	13
WRITET	WRITET	-	15
WRITEU	WRITEU	WRITED	16
WRITEWO	WRITEWORDS	WDIOCOM	42

SIZES

This section describes how to calculate the **approximate** size of the stand alone program files generated by FIXCIN, ROMCIN and UTILCIN. It may often be easier to build a stand alone program and find out how big it actually is than to attempt to calculate the size.

The calculations given below are for programs which include some CINTCODE. There is a saving of 33 words for programs written entirely in assembler.

Basic program size

The basic program size is the size of all the BCPL and assembler code and global definitions. The easiest way to find it is to run PACKCIN on the program and add the 'code size' and 'global data size' figures from the report produced.

It may be estimated from the compiled size of the application code and required SYSLIB sections, subtracting four words for each application hunk and five words for each application SECTION and NEEDS directive.

FIXCIN output file

The file size is the basic program size plus the initialisation overhead (224 words).

Once the file has been loaded and initialisation is complete the initialisation data and code becomes part of the heap. Thus the size of the program when running is the basic program size less two words for each global declared.

ROMCIN output file

The file size is the basic program size plus the initialisation overhead (340 words) plus the length of all the text items defined in the Spec file (ie copyright, title and version texts and '*' command names).

UTILCIN output file

The file size is the basic program size plus the initialisation overhead (400 words) plus the length of all the text items defined in the Spec file (ie copyright, title and version texts and '*' command names) plus an extra two words for each '.ENTRY' command in the Spec file.

STAND ALONE FILE FORMATS

This section describes the formats of the stand alone files generated by FIXCIN, ROMCIN and UTILCIN.

FIXCIN output file

The file is made up of four sections - the entry code, the program, the initialisation data and the initialisation code. The load address and execution address of the file are the same and thus execution begins with the entry code.

The entry code is a few bytes of machine code which load the byte address of the initialisation data into X/Y (low byte in X) then jump to the global SAINIT in the initialisation code. The entry code may be followed by a null byte so that the program is word-aligned in memory.

The program is the CINTCODE and machine code from the **FROM** file stripped of hunk headers, global definitions and relocation hunks. All machine code is relocated.

The initialisation data is a structure made up of a number of words as follows:

- a word of 0 if the stand alone program is written entirely in assembler or of -1 if the program contains CINTCODE;

- the byte address of the global vector;

- the length in words of the global vector;

- a number of 2-word preset data entries terminated by a word of -1;

- the required screen mode, or -1 if the **MODE** argument was not specified;

- the byte address to which the initialisation code should jump. This is normally the start of the interpreter but with all-assembler programs it is the procedure START.

With all-assembler programs this is the end of the initialisation data, but with CINTCODE programs the data continues with:

- byte address of the heap;

- byte address for the end of the heap or -1 if the **HEAPEND** argument was not specified;

- a word normally 0, but -1 if the target hardware is not a BBC Microcomputer.

Each preset data entry consists of a word specifying an even byte address followed by a word containing the data to be written to that address. There is one preset data entry for each global defined in the program (these are used to set up the global vector) plus various others for setting up the interpreter work areas and initialising global data. With all-assembler programs the preset data entries specific to the BCPL interpreter are omitted.

The initialisation code is machine code from the file FIXINI, relocated and stripped of hunk headers, global definitions and relocation hunks.

ROMCIN output file

The file is made up of six sections - the ROM header, the entry code, the program, the initialisation code, the initialisation data and the text table.

The ROM header is structured as follows:

- a 3-byte JMP to the language entry point in the entry code;

- a 3-byte JMP to the service entry point in the entry code;

- the ROM type byte (always C2 hex);

- a byte containing the byte offset from the start of the file to the zero byte following the version string;

- a byte containing the version number;

- the title followed by a zero byte;

- the version text followed by a zero byte;

- the copyright text followed by a zero byte.

The entry code is a few bytes of machine code which jump to the global SAINIT in the initialisation code with the C bit set for a language entry, clear for a service entry. The entry code may be followed by a null byte so that the program is word-aligned in memory.

The program is the CINTCODE and machine code from the **FROM** file stripped of hunk headers, global definitions and relocation hunks. All machine code is relocated.

The initialisation code is machine code from the file ROMINI, relocated and stripped of hunk headers, global definitions and relocation hunks.

The initialisation data is a structure made up of a number of words as follows:

- the byte address of the text table;

- a word of -1 if the stand alone program contains CINTCODE or of 0 otherwise;

- the byte address of the global vector;

- the length in words of the global vector;

- a number of 2-word preset data entries terminated by a word of -1;

- the required screen mode, or -1 if the **MODE** argument was not specified;

- the byte address to which the initialisation code should jump. This is normally the start of the interpreter but with all-assembler programs it is the procedure START.

With all-assembler programs this is the end of the initialisation data, but with CINTCODE programs the data continues with:

- byte address of the heap;

- byte address for the end of the heap or -1 if the **HEAPEND** argument was not specified;

- byte address of the byte beyond the end of the text table, rounded up to a 2-word boundary (used for setting up the heap if running in a 6502 second processor).

Each preset data entry consists of a word specifying an even byte address followed by a word containing the data to be written to that address. There is one preset data entry for each global defined in the program (these are used to set up the global vector) plus various others for setting up the interpreter work areas and initialising global data. With all-assembler programs the preset data entries specific to the BCPL interpreter are omitted.

The text table contains all the names specified by '.ENTRY' commands. It is simply a list of all the names (as a set of characters, not as BCPL strings) with each name terminated by a zero byte and an extra zero byte terminating the table.

UTILCIN output file

The file is very similar to that produced by ROMCIN. The principal differences are:

- the initialisation code is taken from UTILINI;
- the initialisation data is structured differently;
- the text table is followed by an extra structure called the entry point table (aligned on a word boundary).

The initialisation data is a structure made up of a number of words as follows:

- the byte address of the text table;
- the byte address of the entry point table;
- the byte address of the global vector;
- the length in words of the global vector;
- a number of 2-word preset data entries terminated by a word of -1 (these entries set up the global vector and, possibly, the fault routine address in bytes $202/$203).

With all-assembler programs this is the end of the initialisation data, but with CINTCODE programs the data continues with:

- a number of 2-word preset data entries terminated by a word of -1 (these entries set up data used by the interpreter);

- byte address of the heap;

- byte address for the end of the heap or -1 if the **HEAPEND** argument was not specified;

- byte address of the byte beyond the end of the entry point table, rounded up to a 2-word boundary (used for setting up the heap if running in a 6502 second processor);

- byte address of the interpreter entry point.

The text table contains all the names specified by '.ENTRY' commands. It is simply a list of all the names (as a set of characters, not as BCPL strings) with each name terminated by a zero byte and an extra zero byte terminating the table.

The entry point table contains one 3-byte entry for each name in the text table. The first entry corresponds to the first name and so on. The format of each entry is:

```
Byte 0   : bit 7   : 1 = BCPL entry point
                     0 = assembler entry point

           bit 6   : set if screen mode to be set
                     up before program entered

           bits 0-2: required screen mode (only if
                     bit 6 is set)

Bytes 1-2: for a BCPL entry point the global
           number of the BCPL procedure;
           for an assembler entry point the byte
           address of the assembler procedure.
```

SYSTEM DATA AREAS

This section should be read in conjunction with the section 'System data areas' in the Appendix of the development system User Guide. It describes the major differences between the data areas in the development environment and the data areas in the stand alone environment.

The sub-sections used correspond to those in the development system User Guide.

Heap

In the stand alone environment the format of the first two words of each area is:

word 0: address of the next area

word 1: 0 if the area is free
127 if the area is allocated.

Language RAM

Words 0, 2 to 6 and 9 to 10 of the 'Index and miscellaneous data' are not used in the stand alone environment. The area from the despatch routine onwards is used for the main stack (which is not initialised to zeros).

Miscellaneous

There are no miscellaneous system data areas in the stand alone environment.

Zero page

Words 0, 8 to 16 and 19 to 40 are not used in the stand alone environment.

Stacks

When a stack is created in the stand alone environment it is not initialised to zeros.

Store files

There are no store files in the stand alone environment.

Stream control blocks

In the stand alone environment the 'get function' and 'put routine' addresses are undefined if not relevant (eg there is no get function for an output stream).

Device type 7 (store file) is not allowed.

Stream control blocks for current filing system files do not contain the file name.

System index

There is no system index in the stand alone environment.

System saved data

There is no system saved data in the stand alone environment.

SYSTEM PROCEDURES

This section describes the differences between the global procedures used by the stand alone environment and those used by the development environment. It should be read in conjunction with 'System procedures' in the Appendix of the development system User Guide.

The following system procedures are not available in the stand alone environment:

CONTPRG	ENDTRAP	FINDSTFILE
RDTOBLOCK	SFCNTRL	SFSTATE
TRAPSTART	TRUNCVEC	

The system procedures in the stand alone environment are described below.

CHANGECO (32)
Identical to development system version (except that there is no 'end of program trap' facility).

CLOSESTREAM (120)
Identical to development system version.

ENDINT (8)
Called to terminate the program. See 'Program termination' in chapter 6 for more details.

Called by:

ENDINT()

FILENAME (60)
Checks that a string is a valid device name or filename. The procedure NAMESTR (see below) can be used to convert the string to a standard format.

Called by:

dev := FILENAME(string)

dev is -1 if the string is invalid. If the string is valid **dev** is the device code (as held in a Stream Control Block).

IOINIT (7)
Called from SYSINIT (see below) to set up any I/O streams before the application program is entered.

NAMESTR (102)
Copies a file name into a vector, terminating the name with CR and stripping off any leading '/F.'. More precisely if the first character of the name is '/' then the first three characters are ignored.

The resultant vector is not a BCPL string but is suitable for passing to operating system routines eg OSFIND. The name should be checked with FILENAME before calling NAMESTR.

Called by:

vector := NAMESTR(namestring)

The **vector** is taken from the heap and it is the user's responsibility to return it (by FREEVEC) at a suitable time. **vector** is 0 if the GETVEC to get the vector from the heap fails.

STRCNTL (61)

Opens a stream. Called by:

scb := STRCNTL(name, type)

name is the stream name.
type is 1 for input, 2 for output.
scb is the address of the stream control block or 0 for failure.

SYSINIT (119)

The first procedure executed when the BCPL interpreter is started. It performs some initialisation then calls the application program. Called by:

SYSINIT(glbno)

where **glbno** is the global number of the application procedure to be called (1 for START).

BCPL Stand Alone Generator

on the BBC Microcomputer

Acornsoft Limited, Betjeman House, 104 Hills Road,
Cambridge CB2 1LQ, England. Telephone: (0223) 316039

This BCPL system was developed by Richards Computer Products Ltd., a company that specialises in providing BCPL systems. It was developed in collaboration with Dr Martin Richards of the Computing Laboratory, Cambridge University, who invented BCPL in 1967.

SNL12/B