Appendix B - File formats

This appendix describes the format of output files produced by the linker.

B.1 Summary of control file commands

This section is a quick summary of the commands possible in the linker control file.

Lines beginning with * ; or ! are comments and are ignored by the Linker. All letters in the control file input can be in either case and case is not significant.

- **INPUT <file name>**
  Instructs the linker to include all modules from the named file in the link.

- **EXTRACT <module name> FROM <file name>**
  Instructs the linker to find the module named in the file. If the module is found it is included in the link.

- **LIBRARY <file name>**
  Instructs the linker to search the library from start to finish. Any modules in the library which satisfy any currently unresolved references are included in the link.

- **SECTION <section name>**
  Declares a section to the linker. All declared sections are allocated space before any undeclared sections.

- **COMMON <common option>**
  Instructs the linker how to handle COMMON sections (if any are encountered).

- **RELOC <section name>**
  Instructs the linker to collect run time relocation information and place it in the section named.

- **OFFSET <value>**
  Instructs the linker to start address allocation and to write the output file from the address given in the value parameter.

- **DEFINE <symbol> [=] <expression>**
  Defines a symbol at link time. If the expression includes a symbol which has not already been defined then the linker expects to find it in a relocatable object module.

- **DATA <value> [K]**
  Defines the amount of data space required by the program when it is run.

B.2 Relocatable binary format

This section defines the official Sinclair relocatable binary format. It is self-contained and uses some terms with different meanings from those used in the rest of the linker manual.

A relocatable object file consists of a sequence of modules, each of which is a sequence of bytes terminated by an END directive (see below). It should have a Qdos file type of 2 though this will not be enforced by the linker. Interspersed with the sequence of bytes can be directives from the list below; a directive is a sequence of bytes beginning with the hex value FB.

When otherwise unmodified by a directive, a byte indicates that it should be inserted at the current address and the address should be stepped by 1. The special directive FB FB inserts the value FB in this way.

Note that bytes are **overwritten** on (not added into) the byte stream, so that if several sections are located at the same address, it is possible to overlap (or even interleave) their contents. This is useful for Fortran block data.

In the following syntax definition, <words>s and <longword>s need not be word aligned: they just follow on from the preceding data with no padding bytes.

A <string> consists of a length byte (value range 0 – 255), followed by the bytes in the string. A <symbol> is a <string> of up to 32 chars. A symbol should start with a letter (A – Z) or a dot and the other characters may be letters, digits, dollar, underline or dot.

B.2.1 Definition of a SECTION

A SECTION is a contiguous block of code output by the linker. Each section has a name, and any source file can add code to one or more of the sections. A module's contribution to a section is called a subsection.

The linker will arrange that each section or subsection will start on an even address, by inserting one padding byte if necessary. The value of this byte will be undefined.

Note that if a module returns to a section, this is part of the same subsection and the linker will **not** re-align on a word address.

When a section name is used in an XREF command the address of the start of the subsection is used.

Note that section names are maintained separately from symbol names (and module names), so there can be a section, a symbol and a module all with the same name without any danger of confusion.

B.2.2 Directives

B.2.2.1 SOURCE Syntax: FB 01 <string>

The <string> in this directive indicates information about the source code file from which the following bytes were generated. This directive should only appear at the start of a module (ie at the start of the file or immediately after an end directive: see section B.2.3).

The string will start with the **module name** which may be followed by a space followed by a field of further information about such things as the version number or the date of creation or compilation. The string should contain only printable characters and be no longer than 80 characters.

This **module name** should conform to the syntax of a <symbol> defined above, and may be used by the linker to identify individual modules within a library (see section B.2.4). The module name can be generated from a Qdos filename, but if so it is recommended that the Qdos device name is first stripped off.

B.2.2.2 COMMENT Syntax: FB 02 <string>
The <string> in this directive is a line of comment. It will have no effect on the binary file, but should be included at some suitable point in a link map. The string should contain only printable characters and be no longer than 80 characters.

B.2.2.3 ORG Syntax: FB 03 <longword>
This indicates that the bytes following the directive are to start at the absolute address given in the parameter. This applies until the next ORG, SECTION or COMMON directive.

B.2.2.4 SECTION Syntax: FB 04 <id>
This indicates that the bytes following the directive are to be placed in the relocatable section whose name was defined in a DEFINE command with the id value specified. See B.2.2.8.

This applies until the next ORG, SECTION or COMMON directive.

B.2.2.5 OFFSET Syntax: FB 05 <longword>
This directive updates the output address: the longword specifies the address relative to the start of the current subsection or the latest ORG directive.

The parameter is unsigned, so the offset may not be negative.

B.2.2.6 XDEF Syntax: FB 06 <symbol> <longword> <id>
This indicates that the symbol whose name is the <symbol> is defined to be the value given in <longword>, relative to the start of the subsection referred to by the <id>. Note that an <id> of zero defines the symbol to be absolute.

See section B.2.2.8 for definition of <id>

B.2.2.7 XREF
Syntax: FB 07 <longword> <truncation-rule> { <op> <id> } FB

This indicates that the result of an expression involving user symbols or other relocatable elements is to be written into the byte stream. Note that this command does not overwrite some existing bytes, but appends new bytes to the output.

The <longword> parameter defines an absolute term for inclusion in the expression to be evaluated by the linker.

The <truncation-rule> parameter is a byte which defines the size of the final result and the circumstances in which the linker might give a truncation error, or the mode in which truncation should occur (undefined bits must be set to zero). These are the effects of setting each bit:

a If bit 0 is set, the result is one byte.
If bit 1 is set, the result is a word.
If bit 2 is set, the result is a longword.
Only one of these three bits may be set.

b If bit 3 is set, then the number is signed.
See notes below.

c If bit 4 is set, the number is unsigned.
See notes below.

d If bit 5 is set, the reference is PC relative, and the relocated current address (ie the address to be updated by this directive) is to be subtracted before the truncation process.

e If bit 6 is set, runtime relocation is requested (for longwords only). The address of the longword is included in a table generated by the linker which can be used by a runtime loader.

After the <truncation-rule> is a sequence of terms for the expression. <op> is a one-byte operator code and can be 2B for "+" or 2D for "–". <id> is a symbol or section name id as defined in the DEFINE directive (2.8). The special <id> code of 0000 refers to the current location counter (ie the address updated by this directive).

The final FB byte terminates the sequence of terms in the expression.

As an example of the use of the signed/unsigned bits, consider a value which must be written out as a word value; the signed/unsigned bits are interpreted as follows:

resulting value

	<	FFFF8000	always out of range
FFFF8000	–	FFFFFFFF	illegal if 'unsigned' bit is set
00000000	–	00007FFF	always allowed
00008000	–	0000FFFF	illegal if 'signed' bit is set
	>	0000FFFF	out of range

There are some examples of XREF directives in B.2.5 below.

B.2.2.8 DEFINE Syntax: FB 10 <id> <symbol>
FB 10 <id> <section name>

This directive is used in conjunction with XDEF, XREF, SECTION and COMMON.

The directive defines that the <symbol> or <section name> may be referenced by the 2-byte <id> in XREF directives. A <section name> has the same syntax as a <symbol>.

Note that positive nonzero <id> values refer to symbols and negative <id> values refer to section names.

This directive must appear before the <id> value is used in any other directive.

If two <id> values are used to refer to the same symbol, or if one <id> value is reassigned to another symbol the effects are undefined at present.

B.2.2.9 COMMON Syntax: FB 12 <id>
This directive is identical to the SECTION directive except that it informs the linker that the section is to be a common section so that references to this section in **different** object modules refer to the same memory location.

Within the same object module multiple additions to the same section will be appended together as for an ordinary section.

When different modules create common sections of differing size, the linker should create a section equal in size to the largest one.

B.2.2.10 END Syntax: FB 13
This directive marks the end of the current object module. If the file contains only one module, then this will appear at the end of file.

B.2.3 Directive ordering

B.2.3.1 Mandatory Rules

Within a relocatable object file the following rules should be applied to the ordering of the directives within an object module:

a) A SECTION directive (or ORG or COMMON) must appear before any data bytes in the module.

b) A symbol or section's <id> must be defined in a DEFINE directive before it is used in any other directive.

The ordering of other directives is at the discretion of the authors of compilers or relocatable assemblers, though it will normally be dictated by the source code.

B.2.3.2 BNF definition of a relocatable object file

This BNF uses { } to mean 0 or more repetitions of an item.

<relocatable object file> = <module> { <module> }

<module> = SOURCE { <chunk> }END

<chunk> = <header> <body>

<header> = { <header command> } <section command>

<header command> = COMMAND | XDEF | DEFINE

<section command> = SECTION | ORG | COMMON

<body> = { <data byte> | <body command> }

<body command> = OFFSET | XDEF | XREF | DEFINE | COMMENT

B.2.4 Library format

B.2.4.1 Use of libraries in the QL Linker

A library is a relocatable object file as described above, but it will normally contain more than one module. Note that a library can be created by appending smaller libraries or object files.

When the linker processes a LIBRARY command it checks each module to see if it resolves any external references. If so, that module will be included in the link.

The linker also has a facility to extract a specific module from a library, using the module name in the source directive.

B.2.5 Example

The object module format will be illustrated with the aid of this example assembler source module: the file name is "MDV1_EXAMPLE_ASM".

The Program is shown in Fig 1.

```
                                        TITLE       Illustrate the object module format
                                        XDEF        TABLE 1, THIS ROUTINE
                                        XREF        FINAL TAB
                                        XREF        SEARCH TABLE
                                        XREF        THAT ROUTINE, THE OTHER
                                        SECTION     CODE
                                                    :
                                                    :
                                                    :
001234                                  THIS ROUTINE:
001234  317Cxxxx0008                   MOVE.W      # FINAL TAB - TABLE 2,8(A0)
00123A  41FAxxxx                        LEA         TABLE 1(PC),A0
00123E  4EBAxxxx                        JSR         SEARCH TABLE
                                                    :
                                                    :
                                                    :
                                        SECTION     DATA_TABLES
                                                    :
                                                    :
                                                    :
            *note that this assembler interprets "*" as the address
            *at the start of the current line, not the current pc
000072  xxxxxxxxxxxx TABLE 1 : DC.L THIS ROUTINE-*,THAT ROUTINE-*,THE OTHER-*
                                                    :
                                                    :
                                                    :
0000C8  000102030405 TABLE 2: DC.B                    0,1,2,3,4,5,6,7,8,9

    END
```

Fig 1.

The generated object module would then look something like this (in file "MDV1_EXAMPLE_REL"):

```
FB 01 10 45 58 41 4D 50 4C 45 20 32 38 2F 30 39 2F
38 34
SOURCE          EXAMPLE         28/09/84

FB 02 23 49 6C 6C 75 . . . . . .
COMMENT Illustrate . . . . . .

FB 10 FF FF 04 43 4F 44 45
DEFINE -1    CODE

FB 10 FF FE 0B 44 41 54 41 5F 54 41 42 4C 45 53
DEFINE -2    DATA_TABLES

FB 06 06 54 41 42 4C 45 31 00 00 00 72 FF FE
XDEF            TABLE 1   DATA_TABLES

FB 06 0B 54 48 49 53 52 4F 55 54 49 4E 45
00 00 12 34 FF FF
XDEF            THIS ROUTINE                CODE

FB 10 00 01 08 46 49 4E 41 4C 54 41 42
DEFINE        +1        FINAL TAB

FB 10 00 02 0B 53 45 41 52 43 48 54 41 42 4C 45
DEFINE        +2        SEARCH TABLE

FB 10 00 03 0B 54 48 41 54 52 4F 55 54 49 4E 45
DEFINE        +3        THAT ROUTINE

FB 10 00 04 08 54 48 45 4F 54 58 45 52
DEFINE        +4        THE OTHER

FB 02 FF FF
SECTION CODE
```

. . .

```
31 7C FB 07 FF FF FF 38 02 2B 00 01       2D
FF FE          FB 00 08
MOVE XREF -C8 | + FINAL TAB - DATA-TABLES
                    Rules: word

41 FA FB 07 00 00 00 72 2A 2B FF FE FB
LEA XREF     |  + DATA-TABLES
                    Rules:   PC - rel, word, signed

4E CA FB 07 00 00 00 00 2A 2B 00 02 FB
JSR  XREF    |  +SEARCH TABLE
                    Rules: PC-rel, word, signed
. . .

FB 02 FF FE
SECTION DATA-TABLES

. . .

FB 07 00 00 11 C2 04 2B FF FF 2D FF FE FB
XREF 1234-00 72  |  + CODE–DATA-TABLES
Rules: long

FB 07 FF FF FF 8E 04 26 00 03        2D FF FE FB
XREF              -00 72  |  + THAT ROUTINE   - DATA-TABLES
Rules: long

FB 07 FF FF FF 8E 04 2B 00 04        2D FF FE FB
XREF              -0072   |  THE OTHER          + DATA-TABLES
Rules: long

. . .

FB 13
END
```