

David Howells (Original Text)
Norman Dunbar (PDF and corrections)

DBAS Database System

A Guide to using the DBAS database system
on your QL

Dec 2, 2024

Me, Myself, Eye Publishing.

Contents

1	Introduction	1
2	Files	3
2.1	Files Provided	3
2.1.1	Example Databases	4
2.1.2	<i>SuperBASIC</i> Examples	5
2.1.3	Assembler Include Files	5
2.1.4	Assembly Examples	5
3	DBAS System Organization	7
3.1	File Organization	7
3.2	Memory Allocation per Database	8
3.3	Operating System Interface	9
3.3.1	SuperBASIC Interface	9
4	Database Machine Code Interface	11
4.1	Loading the Vectors	11
4.2	Using the DBAS Trap	11
4.2.1	The FS.DBASE Trap	12
4.3	The DBAS Vectors	13
4.4	Memory Allocation	13
4.4.1	Allocate Memory	14
4.4.2	Release Memory	14
4.5	DBAS Vectors	15
4.5.1	Create a database	16
4.5.2	Open Database	18
4.5.3	Goto Record	19
4.5.4	Fetch Field	20
4.5.5	Put Field	22
4.5.6	Write Record	24
4.5.7	Update Record	25

4.5.8	Delete Record	26
4.5.9	Database Information	27
4.5.10	Order Database	28
4.5.11	Select/Unselect Records	29
4.5.12	Reset Selection	31
4.5.13	Find Record	32
4.5.14	Ordering Information	34
4.5.15	Close Database	35
4.5.16	Import Database	36
4.5.17	Export Database	38
4.5.18	Update Extra Information	40
4.5.19	Fetch Extra Information	41
4.5.20	Set String Compare Tables	42
4.5.21	Compare Values	43
4.5.22	Open Directory	44
4.5.23	Load Code Section	45
4.5.24	Database Space	46
4.5.25	Save Code section	47
4.5.26	Get Field Name Vectors	48
4.5.27	Get Subfield Vectors	49
4.5.28	Get Extra Vectors	50
5	DBAS SuperBASIC Interface	51
5.1	Loading the <i>SuperBASIC</i> Interface	51
5.2	Symbols Used	51
5.3	General	52
5.4	Memory Allocation	52
5.4.1	Database Blocks	53
5.5	Field Names	53
5.6	Array Parameters	54
5.6.1	Procedures Summary	54
5.6.2	Functions Summary	55
5.7	Procedures	56
5.7.1	Add a Field	56
5.7.2	Add a Record	56
5.7.3	Copy Database to New Database	56
5.7.4	Create Database	56
5.7.5	Close Database	57
5.7.6	Exclude Records	57
5.7.7	Export Database	57
5.7.8	Find Records	58
5.7.9	Import Database	58
5.7.10	Select Records	59
5.7.11	Load Field Names	60
5.7.12	Search Ordered Records	61

5.7.13	Open Database	61
5.7.14	Open Device Directory	61
5.7.15	Order Database	62
5.7.16	Delete Record	62
5.7.17	Remove Field	62
5.7.18	Remove Filed Name Memory	63
5.7.19	Select All Records	63
5.7.20	Position Record Pointer	63
5.7.21	Save Name List	63
5.7.22	String Comparison Order	63
5.7.23	Search Records	64
5.7.24	Set Field Value	64
5.7.25	Save Extra Information	65
5.7.26	Order Key Size	65
5.7.27	Rename Field	65
5.7.28	Add Subfields	66
5.7.29	Remove Subfields	66
5.7.30	Replace Subfield Content	66
5.7.31	Update Record	66
5.8	Functions	67
5.8.1	Count Selected Records	67
5.8.2	Database Control Block Address	67
5.8.3	Fetch Field	67
5.8.4	Fetch Extra Information String	67
5.8.5	Field Length	67
5.8.6	Field Name	68
5.8.7	Field Number	68
5.8.8	Field Count	68
5.8.9	Field Type	68
5.8.10	Error Functions	68
5.8.11	Did Search Succeed	68
5.8.12	Return Current Record Number	69
5.8.13	Get Subfield Contents	69
5.8.14	Subfield Numbers	69
6	C68 Library	71
6.1	Structures and Unions	71
6.1.1	Field List Structure	71
6.1.2	Parameter Definitions	72
6.2	Library Functions	72
6.2.1	Add Field	72
6.2.2	Add Record	72
6.2.3	Close Database	73
6.3	Global Variables	79

7	Field Name Handling	81
7.1	Usage	81
7.2	File Format	81
7.3	Memory Format	82
7.4	<i>SuperBASIC</i>	82
7.5	Machine Code	82
7.5.1	The Main Vector	82
7.5.2	Extra Vectors	83
8	String Comparison Tables	87
8.1	Introduction	87
8.2	<i>SuperBASIC</i> Usage	87
8.3	Machine Code Usage	88
8.4	The Tables	89
8.5	Modifying DATA_BIN and DROM_BIN Tables	89
9	Directory Support Using Database Vectors	91
9.1	Associated Files	91
9.2	Usage	91
9.2.1	Fields	91
9.2.2	Records	92
9.2.3	Other Access	92
10	Manipulate A Database	95
10.1	Use From <i>SuperBASIC</i>	95
10.2	Use From Machine Code or C	96
10.3	Source files for ALTER_BIN	96
	Appendices	99
A	Installation of Printer Drivers	101
A.1	Associated Files	101
A.2	Usage	101
A.3	Valid Items	102
A.4	Source Files	102
B	Archive Replacement	103
B.1	History	103
B.2	Overview	103
B.2.1	General	103
B.2.2	Superbasic	104
B.2.3	Machine code	104
B.2.4	C68 facilities	104
B.3	Implementation	105

C	Updates	107
C.1	Changes to Database Handling	107
C.1.1	Version 2.00	107
C.1.2	Version 2.01	107
C.1.3	Version 2.02	107
C.1.4	Version 2.03	108
C.1.5	Version 2.10	108
C.1.6	Version 2.11	109
C.1.7	Version 2.12	109
C.2	Changes to DBPTR_BIN	109
C.2.1	Version 1.01	109
C.2.2	Version 1.02	109
C.2.3	Version 1.03	110

Chapter 1

Introduction

28, Medlock Crescent,
Handsworth,
Sheffield.
S13 9BD

After programming extensively in *Archive*, I decided to write a database package that could be used directly from *SuperBasic* as I don't particularly like the programming language that *Archive* uses.

After a number of false starts, I decided to write the package to be used from machine code, with an interface program to allow it to be used from *SuperBasic* as well.

The machine code part was to be used entirely through the TRAP #3 system originally, and not through vectors at all. Unfortunately, this caused strange events to occur within the RAM-disk and Floppy-disk drivers, which ended up crashing the computer. So I decided to only have one trap, whose function was to return the vector base address and Database version number.

Fortunately, this arrangement didn't cause any bother with the device drivers. All the main routines are now accessed relative to this vector base address.

Unfortunately, some parts of the *Archive* programming language are impossible to implement in exactly the same way in *SuperBasic*, a good example being:

```
select a=1 and b=c
```

With Basic this cannot be used for two reasons. Firstly `select` is already used by basic, and secondly `a=1 and b=c` is processed by the ROM before being passed to `DBAS_BIN`.

Suggested chapter reading order:

1. Chapter 2 - **Files** all about the supplied files on the distribution disc. (Originally FILELIST_DOC)
2. Chapter 3 - **DBAS System Organization**. (Originally ORGANIZE_DOC)
3. Chapter 5 - **DBAS SuperBASIC Interface**. (Originally DBASREF_DOC) and associated example programs

4. Chapter 4 - **Database Machine Code Interface**. (Originally DATAREF_DOC) and associated example programs
5. And then, anything else that takes your fancy!

The sources for the database editing programs will go on another disk.
Enjoy using the package.

David W. Howells

Chapter 2

Files

There are a number of files supplied as part of the *DBAS* package. These are documented next.

2.1 Files Provided

DATA_BIN The most important file. It contains the machine code part of the database system. It must only be loaded by Job 0 (*SuperBASIC*), and must be loaded before **DBAS_BIN** can be used.

DBAS_BIN The *SuperBASIC* part of the database handler (not required if the package is being used directly from machine code). If you have *Minerva*, this can be loaded in by a *MultiBASIC*, which would only extend that particular Basic, and no other.

BOOT A boot file for loading **DATA_BIN** and **DBAS_BIN**.

CLCHP_BIN A The **CLCHP** procedure for releasing memory accidentally left after a **CLOSE**. To load it:

```
a=RESPR(70): LBYTES FLP1_CLCHP_BIN,a: CALL a
```

Or, if you have Toolkit 2:

```
LRESPR FLP1_CLCHP_BIN
```

See Chapter 5 - **DBAS SuperBASIC Interface** for details.

TOGGLE_THING Activate/Deactivate *Hotkey II* type **THING** adding routine in **DATA_BIN/DROM_BIN**. (BASIC program)

CODE_SAVE_BIN A program to set the code section of a database. Just **EXECute** and follow the instructions.

TEST_MC4_BIN Utility. See below.

ALTER_BIN Utility. See below.

DBPTR_BIN Utility. See below.

Note

The *Quill* documents mentioned in this PDF have, in the most part, been used to create this PDF file. They remain mentioned here so as to advise anyone, finding them on the distribution disc, as to their original purpose. *ND 02/12/2024*

The following are *QUILL* document manuals for use of the system:

ARTICLE_doc An article written for QUANTA magazine. See Appendix **B - Archive Replacement**.

INTRO_doc The introduction document. See Chapter **1 - Introduction**.

CHANGES_doc The changes made to V2.00 and beyond. See Appendix **C - Updates**.

ORGANIZE_doc This is a document showing the organization of a database file. It also has some information about the organization of the memory blocks used by the system (most is about the machine code part). See Chapter **3 - DBAS System Organization**.

DATAREF_doc The manual detailing the machine code routines. See Chapter **4 - Database Machine Code Interface**.

DBASREF_doc The manual detailing the BASIC functions and procedures. See Chapter **5 - DBAS SuperBASIC Interface**.

SCPTR_doc This is a manual detailing the use of, and how to modify string compare tables. See Chapter **8 - String Comparison Tables**.

DDIR_doc Further details of the use of the database package handling directories. See Chapter **9 - Directory Support Using Database Vectors**.

CODE_doc Further details of the use of the code section. See Chapter **?? - ??**.

NAMES_doc Further details of the name handling. (Including vector descriptions). See Chapter **7 - Field Name Handling**.

XVECTORS_doc Further details of the extra vectors. See Chapter **7**, Subsection **7.5.2 - Extra Vectors**. *Note: XVECTORS_DOC was not on the distribution disc. ND 02/12/2024*

SUBFIELD_doc Further details of the subfield handling vectors. See Chapter **5**, Subsection **5.8.13 - Get Subfield Contents**. *Note: SUBFIELD_DOC was not on the distribution disc. ND 02/12/2024*

2.1.1 Example Databases

The following are example Databases; use the ALTER_BIN utility for easy viewing. See Chapter **10 - Manipulate A Database** for details of the utility.

GAZET_DBS Catalogue of countries in the world (out of date) and other relevant information. All string fields variable length type.

INSTALL_DBS Installation data for PSION printer drivers. (Eg: QUILL). See Appendix **A - Installation of Printer Drivers**.

2.1.2 *SuperBASIC* Examples

The following are example *SuperBASIC* programs:

`DATA_DUMP` Dump out the control portion of a database file. See [3 - DBAS System Organization](#) for an explanation of the meaning of the bits - requires *Toolkit* 2 hex functions.

`DDIR_BAS` Demonstration of the use of the database package in handling directories.

`TEST_BASIC1` Demonstration of most of the facilities available from the database system.

`TEST_BASIC2` Demonstration of the IMPORT/EXPORT facilities.

`TEST_BASIC3` Demonstration of the find facilities.

`TEST_BASIC4` Demonstration of CREATE with array-type parameter passing.

`TEST_BASIC5` Demonstration of IMPORT with array-type parameter passing.

`TEST_BASIC6` Demonstration of EXPORT with array-type parameter passing.

2.1.3 Assembler Include Files

The following contain lists of label definitions suitable for inclusion in assembler programs:

`DATA_IN` The database memory offsets file. These offsets are the layouts of memory blocks. It also contains some equates for configuring the system at linking.

`QDOS_DATA_IN` The database access offsets file. These are the offsets used for accessing the machine code parts of the system, and also for laying out memory blocks used for passing parameters.

2.1.4 Assembly Examples

The following are example Machine code programs and relevant equate files:

`TEST_MC1_ASM` and `TEST_MC1_BIN` Demonstration of using `FSD.CRT` from machine code, plus various other routines. Just EXECute.

`TEST_MC2_ASM` and `TEST_MC2_BIN` Demonstration of `FSD.EXPT` from machine code. Just EXECute.

`TEST_MC3_ASM` and `TEST_MC3_BIN` Demonstration of `FSD.SEL` from machine code, with both internal-parameter, and user-routine type calls. `FSD.FIND` and `FSD.CONT` with `D1=FSDF.SEL` uses the same format of parameters.

`TEST_MC4_ASM` and `TEST_MC4_BIN` Demonstration of `TRAP #4` used with the database routines. Print out the contents of the current record.

To load the extension, use:

```
a=RESPR(1024): LBYTES FLP1_TEST_MC4_BIN,a: CALL a
```

or, with *Toolkit 2*:

```
LRESPR FLP1_TEST_MC4_BIN
```

Then:

```
DISPLAY #database_chan TO #output_chan
```

For example:

```
DISPLAY #3 TO #1
```

ALTER_doc and ALTER_BIN A program to edit the contents of a database. Just EXECute. The source will be on another disk.

DBPTR_doc, DBPTR_HELP_DBS and DBPTR_BIN A program to edit the contents of a database with the support of the QJump pointer environment. Requires ptr_gen and wman; and hot_rext is useful Just EXECute. The source will be on another disk.

INSTALL_doc, INSTALL_DBS, INST_BAS and INST_BIN Demonstration of code section usage. (Use CODE_SAVE_BIN to store it in INSTALL_DBS.)

INST_IN, INST_LINK and INST_xxxx_ASM The source files for INST_BIN.

FAST_MATH_ASM 32-bit multiply and divide subroutines.

FAST_FLIN_ASM A subroutine to float a long integer.

QDOS_TRAP_ASM QDOS TRAP equate file.

QDOS_VECT_ASM QDOS Vectors equate file.

QDOS_SCRN_ASM QDOS Screen TRAP equate file.

The following are example C (C68) files and associated support files, and documentation.

DBC_c DBC_bin Example C source file; to demonstrate fsd_incl, fsd_excl, fsd_srch, and fsd_srch_c record "test" routine handling.

INCLUDE_database_h Header file for database C library. Problems may arise due to prototype definitions being invalid for certain compilers & C68 versions; see file.

LIB_libdbas_a Database C library using _oserr & _dberr for older C68 versions.

LIB_libdbasdu_a Database C library using __oserr & __dberr for more recent C68 versions.

LIBDBAS_doc Documentation covering use of the C library routines, and the structures defined in the include file.

LIBdbas_source_arc Collection of source files that make up the database C library. Compressed and collected using the arc program supplied with C68, (this may be available from the QUANTA C librarian).

Chapter 3

DBAS System Organization

3.1 File Organization

The database file file begins with a header, the structure of which is documented in Table 3.1.

Table 3.1 Database file header

Offset	Size	Description
\$00	Long	'DBAS' identifier.
\$04	Byte	Database flags: bit 0: =1 - Dynamic records. bits 1-7: Version, currently 1
\$05	Byte	Reserved - currently set to 0.
\$06	Word	Record pointer - This is a pointer to where the data begins, 0 if a directory type database.
\$08	Word	Maximum record length - This is the length of a fixed record.
\$0A	Word	Record count - Total number of records held in file, unaffected by FSD .SEL, INCLUDE, or EXCLUDE calls.
\$0C	Word	Field count - Total number of fields held in each record.
\$0E	Long	Record length table offset (Dynamic records only).
\$12	Word	Length of reserved section 0.
\$14	Word	Length of reserved section 1.
\$16	Word	Length of reserved section 2.
\$18	Word	Length of code section.
\$1A	...	Field definitions. 8 bytes per field definition—see Table 3.2 on Page 8.
...	Bytes	Reserved Section 0. For size, see offset \$12.
...	Bytes	Reserved Section 1. For size, see offset \$14.
...	Bytes	Reserved Section 2. For size, see offset \$16.
...	Word	Length of extra information data, beginning with the field name list.
...	Bytes	Field names list and extra information—see SEXTRA or FSD .XTRS.
...	Bytes	Code section—see ?? - ??.

The field definitions, beginning at offset \$1A, consist of 8 bytes per field. Table 3.2 shows the structure of each of the field definition table entries.

Table 3.2 Database field details

Offset	Size	Description
\$00	Word	Field offset, n/a if dynamic records.
\$02	Byte	Flags; bit 7: Set=Dynamic (variable length) field
\$03	Byte	Field type.
\$04	Word	Field length; for strings = maximum data length + (1 for count byte or 2 for count word).
\$06	Word	Reserved

Directly after the code section, and pointed to by the word at offset \$06 of the file, the records are stored.

Finally, if the records are dynamic, we have the record length table—pointed to by the long word at offset \$0E of the file. This is a simple table consisting of a single word giving the length of the entire record, one for each record in the database file.

3.2 Memory Allocation per Database

All subsequent memory blocks belonging to a database are allocated with the job ID of the owner of the control block.

The first two blocks of memory are always in existence per open database:

1. The control block: the format of this is described in file DATA_IN. It is used to reference all other blocks of memory, and to remember channel ID, current record position, field information, order information, etc. The Database ID returned by FSD.CRT and FSD.OPEN is the address of the control block.
2. The current record block: This is a store of the current record or a new record. Fields within it are placed and retrieved with FSD.PUT and FSD.GET. This is added to/overlaid on/retrieved from the file with FSD.APPN, FSD.UPDT, FSD.POSA and FSD.POSR.
3. The record length buffer: This is a block holding a section of the record length table stored at the end of a file. This buffer and the record length table are only in existence for a database containing dynamic records. The size of the buffer can be configured to be determined automatically, or can be set to a fixed size.
4. The order control block: This is a store of record offsets in order if FSD.ORDR has been issued, with bit 15 of each word set if the record has been deselected if FSD.SEL has been issued. (Note: FSD.SEL can be used even if FSD.ORDR has not been issued, in which case an order control block is allocated if one does not already exist).

5. The order sort block: This is a temporary block which is only exists during `FSD.ORDR` for key field storage.
6. The name table: This is a store of offsets and lengths of fields stored in the name list, plus a user defined long word per field—see `DATA_IN`.
7. The name list: This is a store for the first part of the extra information (up to the first LF), that defines the field names. It is kept in the same format as in the file.
8. Pieces of memory is temporarily allocated when large parts of the file need to be moved around or accessed, eg: `UPDATE`, `REMOVE`, `SEXTRA`, `IMPORT`, `LOAD_NAMES`, `SAVE_NAMES`, `ADD_FIELD`, `REMOVE_FIELD`, but not `APPEND`. `FSD.UPDT`, `FSD.DEL`, `FSD.XTRS`, `FSD.IMPT`, `FSD.NMLD`, `FSD.NMSV`, `FSD.ADDF`, `FSD.REMF`, but not `FSD.APPN`.
9. A job is created by `IMPORT` and `EXPORT` to allow the use of A6-relative vectors.

3.3 Operating System Interface

Each Directory Device Driver implemented at the time `DATA_BIN` is CALLED, has its IO pointer pointing to a piece of common heap called a physical device record (one per Directory Device Driver). This contains a JSR to the database trap code, followed by a JMP to the Directory Device Driver's IO routine, plus a couple of pointers, the second of which is currently unused—see `DATA_IN`.

If the `THING` system is implemented (eg: QJUMP's Hotkey System II v2.03 onwards) then a utility type thing called `DATABASE` can be added (The facility must first be activated with `TOGGLE_THING`). Directly following the 8 byte header is the vector base address relative to the base of the header—see Chapter 4 - Database Machine Code Interface, `TRAP #3` with `D0=FS.DBASE`).

3.3.1 SuperBASIC Interface

Each *SuperBASIC* channel block (ie: the memory for #0, #1, #2...) has the channel ID long word at \$00, the `FOUND` return byte stored at \$17, and the Database ID long word stored at \$24. Information about turtle and relative graphics are stored in between.

Each channel number (#ch) is therefore a reference to both the channel ID and the database ID.

This block is allocated inside the basic area pointed to by `bv.chbas` (A6) and `bv.chp` (A6). `OPEN`, `OPEN_IN`, `OPEN_NEW` also allocate blocks inside this area.

All other database memory is allocated in the common heap, and is chained in a linked list pointed to by `$E0(A6)` of the basic area. Each further pointer is stored at the start of the common heap block, followed by "DATABASE". This method is compatible with Toolkit II's `ALCHP`, `RECHP`, and `CLCHP`.

If a *SuperBASIC* hands a database ID to another job (such as `DBPTR_BIN`), then any allocated memory will automatically be attached/removed from the owning *SuperBASIC*'s `ALCHP` list.

No variables are allocated by the `CREATE` or `OPEN_DATA` procedures unlike in archive.

Variable names are stored in the `SEXTRA` string and can be accessed when loaded—see Chapter 5 - *DBAS SuperBASIC Interface* under field names and `LOAD_NAMES`, `FLNAME`, `FLFNAME`, `SLNAME`, `SAVE_NAMES`, `REMOVE_NAMES`.

Chapter 4

Database Machine Code Interface

4.1 Loading the Vectors

The machine code vectors make up DATA_BIN.

DATA_BIN should be loaded into the resident procedure area, using something like:

```
a=RESPR(20*1024)
LBYTES flp2_DATA_BIN, a
```

or

```
LRESPR flp2_DATA_BIN
```

The vectors can be configured using the QJUMP *Config* program. These things can be changed:

- Default order key length
- Default compare string case dependence
- Database thing activation
- Record length buffer size

Note that all memory is allocated as to be owned by the job which opened/created/imported the database. See Section 4.4 - **Memory Allocation**, on Page 13, for an explanation of MEMORY ALLOC.

4.2 Using the DBAS Trap

To find out the database version number and the vector base address (pointer to list of vectors), a trap must be used.

The trap is only available on Directory Device Drivers—these are devices which can produce a DIR listing from *SuperBASIC*—which were linked in when the CALL

procedure or the `DATA_EXT` procedure was issued. So make sure all the Device Drivers that you are going to use are installed before the package is `CALLed` or `DATA_EXTed`.

Directory Device Drivers include things like MDV, FLP, WIN, RAM, and N, but not NET, PIPE, CON, SCR, SER, and PAR.

To use the package with the network file server, any machine that wishes to use it, must have the package in memory, or a crash will certainly occur. Eg:

- QL net no. 1 has `DATA_BIN` in memory, and a hard disk.
- QL net no. 2 has neither.
- If QL 2 accesses the trap (see below) on QL 1's hard disk, then the operation *may* be successful. If it then tries to access vectors at the address returned, a crash will almost certainly occur.
- However if QL 2 has `DATA_BIN` in memory, and QL 1 has or has not, then:
- All accesses by QL 2 to the package will not crash, provided no duff values (eg: Database ID) are supplied.

4.2.1 The `FS.DBASE` Trap

Call this trap as the base address is required to use any of the database vectors.

FS.DBASE TRAP #3 D0=\$44		
Call Parameters		Return Parameters
D1		D1.L ASCII version number
D2		D2.L Preserved
D3.W Timeout		D3.L Preserved
A0.L Channel ID	A0.L Preserved	
A1		A1.L Vector base address
A2		A2.L Preserved
Error return:		
ERR.BP - Not implemented		
ERR.NO - Channel not open		
ERR.NC - Not complete		
ERR.NI - Not implemented		

The vector base address should never change, so it should be safe to do this trap once only.

4.3 The DBAS Vectors

All the vectors are referenced relative to the vector base address, for example, assuming register A2 holds the database vector base address:

```
JSR    FSD.CRT(A2)
```

If parameter pointers on a call have to be used relative to A6, as *SuperBASIC* requires, then use:

```
TRAP   #4
JSR    FSD.xxxx(An)
```

Note: this does not include the address of the memory allocation routines, which are code and are *always* treated as absolute addresses regardless of whether a TRAP #4 has been issued or not.

Note

If a parameter register is written in the format `xxxx.W, yyyy.W` then `xxxx` is in the high word and `yyyy` in the low word.

All registers unused for parameter passing/returning are preserved if in range D1–D6 and A0–A5, if they are used by the routines internally. D7 and A6 are treated as volatile and are free to change without upsetting anything (as *SuperBASIC* requires with A6).

D0 is always returned with 0 for OK or an error code otherwise. The Z flag in the Status register is always returned set according to the value in D0, and so is set when no error has occurred. The error codes given in the descriptions do not include IO or memory allocation errors.

4.4 Memory Allocation

The database system requires blocks of memory to store information in. These must not move during calls to the database system, and therefore cannot be part of the *SuperBASIC* area which has a habit of moving at random.

An address (referred to as MEMORY ALLOC) is supplied to any vectors which make or open a database, eg: `FSD.CRT`, `FSD.OPEN`, `FSD.DDIR`, and `FSD.IMPT`. This address is used for all subsequent memory transactions.

This address can be 0—in which case `MT.ALCHP` and `MT.RECHP` are used directly—or the address of a piece of user-written code to allocate (`address+0`); and release (`address+6`) memory can be supplied.

Note

The Job ID of the owner is handed over, and the allocated memory must belong to that job!

4.4.1 Allocate Memory

The user supplied code to at *MEMORY_ALLOC* + 0 to allocate DBAS memory must use the following parameters.

FSD.OPEN		JSR \$18(An)	
Call Parameters		Return Parameters	
D0		D0.L	Error code
D1.L	Amount required	D1.L	Undefined
D2.L	Owner Job ID	D2.L	Undefined
D3		D3.L	Undefined
A0		A0.L	Address of memory
A1		A1.L	Undefined
A2		A2.L	Undefined
A3		A3.L	Undefined

4.4.2 Release Memory

The user supplied code to at *MEMORY_ALLOC* + 6 to release DBAS memory must use the following parameters.

FSD.OPEN		JSR \$18(An)	
Call Parameters		Return Parameters	
D0		D0.L	Error code
D1		D1.L	Undefined
D2.L	Owner Job ID	D2.L	Undefined
D3		D3.L	Undefined
A0.L	Address of memory	A0.L	Undefined
A1		A1.L	Undefined
A2		A2.L	Undefined
A3		A3.L	Undefined

4.5 DBAS Vectors

The following DBAS vectors are provided.

- FSD.CRT \$14 Create database
- FSD.OPEN \$18 Open database
- FSD.POSA \$1C Position absolute record pointer
- FSD.POSR \$20 Position relative record pointer
- FSD.GET \$24 Get record
- FSD.PUT \$28 Put record
- FSD.APPN \$2C Append record
- FSD.UPDT \$30 Update record
- FSD.DEL \$34 Delete record
- FSD.INFO \$38 Get information
- FSD.ORDR \$3C Order database
- FSD.SEL \$40 (De)select records
- FSD.RES \$44 Reset record selection
- FSD.FIND \$48 Find record
- FSD.CONT \$4C Continue find
- FSD.OINF \$50 Get order information
- FSD.CLOS \$54 Close database
- FSD.IMPT \$58 Import database
- FSD.EXPT \$5C Export database
- FSD.XTRS \$60 Save extra information
- FSD.XTRL \$64 Load extra information
- FSD.SCPT \$68 Set string compare table pointer. See Chapter 8 - [String Comparison Tables](#).
- FSD.COMP \$6C Compare two strings/integers/floating point numbers
- FSD.DDIR \$70 Open database for a directory channel. See Chapter 9 - [Directory Support Using Database Vectors](#).
- FSD.CDLD \$74 Load code section
- FSD.UPF \$78 Make space in file
- FSD.DNF \$7C Remove space from file
- FSD.CDSV \$80 Save code section
- FSD.NAME \$84 Get ptr to name handling vectors See Chapter 7 - [Field Name Handling](#).
- FSD.SUBF \$88 Get ptr to subfield handling vectors.
- FSD.XVEC \$90 Get ptr to extra vectors.

4.5.1 Create a database

A channel to a named file must be opened before use—this will become the new database file. However, it is *not closed* by any of the vectors, unlike the *SuperBASIC* procedure `CLOSE_DATA`, which closes both the database and the channel.

FSD.CRT		JSR \$14(An)	
Call Parameters		Return Parameters	
D1		D1.L	0.W, field no.W
D2.B	No. of fields	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Channel ID	A0.L	Database ID
A1.L	Pointer to field list	A1.L	Preserved
A2.L	MEMORY ALLOC	A2.L	Preserved
Error return:			
ERR.OV - Record too big			

The channel supplied to this vector can be `IO.OLD` (0) (previous contents erased), `IO.NEW` (2) or `IO.OVERW` (3), but must not be `IO.SHARE` (1), or `IO.DIR` (4) (see the `IO.OPEN` trap, register D3).

In the field list, each field must be represented by a pair of 16 bit words:

- Word 0 - Field type
 - 0 = String,
 - 1 = Integer.W,
 - 2 = Integer.L,
 - 3 = Floating point)
- Word 1 - Field length;

The field length varies depending on the field type. Permitted values are:

- +ve - Fixed length string: maximum length excluding count.
- -ve - Variable length string: maximum length excluding count.
- 0 - For numeric fields, the length is ignored but should be zero.

The 2 words for field 1 must be pointed to by 0(A1) or 0(A6,A1.L); field 2 must lie at 4(A1) or 4(A6,A1.L), etc. The addition of A6 depends on a preceding `TRAP #4` being issued.

On creation of a new database, an “extra information” string is created, giving a *blank field name list* which will take the following format:

"", "", . . . , ""<CR><LF>

See Section 4.4 - **Memory Allocation**, on Page 13, for an explanation of MEMORY ALLOC.

See Chapter 7 - **Field Name Handling**, on Page 81, for an explanation of field names and how to set them.

If an error occurs, then all allocated memory is released before returning.

4.5.2 Open Database

A channel must be opened to the database file before using this vector. As before, it is not closed by any of the vectors, and must be closed manually with `FSD.CLOS` followed by `CLOSE #n`.

FSD.OPEN		JSR \$18(An)	
Call Parameters		Return Parameters	
D1		D1.L	Record count.W, field count.W
A0.L	Channel ID	A0.L	Database ID
A2.L	MEMORY ALLOC	A2.L	Preserved
Error return:			
ERR.ND - B Not a Database (application defined error code)			

The channel supplied to this vector can be `IO.OLD` (0) or `IO.SHARE` (1). It cannot be `IO.NEW` (2), `IO.OVERW` (3), or `IO.DIR` (4) (See the `IO.OPEN` trap, register D3).

The value returned in `D1.L` consists of two separate word values; The count of the number of records in the database in the high word and the number of fields in each record in the low word.

See Section 4.4 - **Memory Allocation**, on Page 13, for an explanation of `MEMORY ALLOC`.

If an error occurs, then all allocated memory is released before returning.

4.5.3 Goto Record

Position the current record pointer to a specific record, and fetch that record from the file.

FSD.POSA		JSR \$1C(An) Position Absolute	
FSD.POSR		JSR \$20(An) Position Relative	
Call Parameters		Return Parameters	
D1.W	Record pointer	D1.W	New record pointer
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.NF - No records.			

The current record will be the requested record if in range, else it will be the nearest record (the pointer is unsigned).

The current record buffer is returned holding the current record.

Only records which are selected can be pointed to, deselected records are totally ignored, ie: D1 = 0: 1st selected record D1 = 1: 2nd selected record ...

The records are pointed to in order if one imposed.

4.5.4 Fetch Field

Get a field from the current record buffer. Fields number from 1, not zero.

FSD.GET		JSR \$24(An)	
Call Parameters		Return Parameters	
Option 1:			
D1.B	Field Number	D1.W	Length of data returned
D2.W	Destination length	D2.L	Preserved
D3		D3.L	Preserved
Option 2:			
D1.B	Field Number	D1.W	Length to move
D2.W	0	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Destination pointer	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR. OR - Field not found			
ERR. BO - Destination full (partial data copied)			

- With option 1, field data is read into the buffer, and D1.W is set to hold the data size.
- With option 2, field data is *not* copied into the buffer, only the length is returned in D1.W. It does not return the buffer size as does FSD.INFO. The data length in D1.W *excludes* space for a string count word.

If a string field is returned, the length word is *not* placed in the buffer.

4.5.4.1 Example

The code to return the data in field four, from the current record in the database, into a buffer would resemble the following. The data size word is stored at the start of the buffer on a successful read.

```
; Assumes Database ID is in D0 and Dbas vector is in A5.
moveq    #4,d1                ; Field number four.
move.w   #BFR_SIZE,D2         ; Buffer size, maximum.
lea      buffer+2,a1           ; Pointer to data buffer.
jsr      fsd.get(a5)           ; Fetch field 4's data.
bne      errorHandler         ; Oops!
move.w   d1,-2(a1)             ; Store the size word.
```

```
    ...  
buffer  
    ds.w    1  
    ds.b    BFR_SIZE
```

4.5.5 Put Field

Place a field into the current record buffer. This does not write to the database, only to the buffer. You must write the record as a new one with `FSD . APPN.` or to update the current record, call `FSD . UPDT.` Fields number from 1, not zero.

FSD.PUT		JSR \$28(An)	
Call Parameters		Return Parameters	
D1.B	Field number	D1.L	Undefined
D2.W	Data length	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Pointer to data	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR . OR - Field number out of range			
ERR . BO - Data too big (no data copied)			

- The buffer must be *data only*, the data *should not* include a length count.
- For a word integer, the data size will be two.
- For a long integer, the data size will be four.
- For a floating point value, the data size will be six.
- For a string, the data size will be the length of the data, not including the size word.

4.5.5.1 Example

The code to update the data in field two, in the current record in the database, from a buffer would resemble the following. The data size word is stored at the start of the buffer in the normal QDOSMSQ string format.

```

; Assumes Database ID is in D0 and Dbas vector is in A5.
moveq    #2,d1                ; Field number two.
lea      buffer,a1             ; Pointer to data buffer.
move.w   (a1)+,d2              ; Data size.
jsr      fsd.put(a5)           ; Write field 4's data.
bne      errorHandler         ; Oops!
...

buffer
dc.w     bufferEnd-buffer-2
dc.b     'New Data'
bufferEnd
equ      *
```

- The buffer must be *data only*, the data *should not* include a length count.
- For a word integer, the data size will be two.
- For a long integer, the data size will be four.
- For a floating point value, the data size will be six.
- For a string, the data size will be the length of the data, not including the size word.
- The data are only written to the record buffer. The record buffer must be written back to the database as a new record (`fsd.append`), or, as an update to the current record (`fsd.updt`).

4.5.6 Write Record

Adds the contents of the current record buffer, to the database file as a new record.

FSD.APPN		JSR \$2C(An)	
Call Parameters		Return Parameters	
D1		D1.W	Record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.OV - Too many records			

The internal buffer must hold the contents of the new record on calling.

The record count returned is the count of selected records only. Note: the new record is automatically selected.

The database is reordered when this vector is called, if an order is already imposed.

4.5.7 Update Record

Updates the current record in the file with the contents of the current record buffer.

FSD.UPDT		JSR \$30(An)	
Call Parameters		Return Parameters	
D1		D1.W	Record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR . EF - No current record			

The internal buffer must hold the contents of the new record.

The record to be replaced is the current one, as set by FSD . POSA, FSD . POSR etc.

The record count returned is of selected records only. Note: the record remains selected.

The database is reordered when this vector is called, if an order is already imposed.

4.5.8 Delete Record

Delete a record from the file.

FSD.DEL		JSR \$34(An)	
Call Parameters		Return Parameters	
D1		D1.W	Record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR . EF - No current record			

The records will be shuffled down and the file truncated when a record has been deleted.

The record to be deleted is the current one, as set by FSD . POSA, FSD . POSR etc.

The record count returned is that of selected records only.

The database is reordered when this vector is called, if an order is already imposed.

4.5.9 Database Information

Gets information about the database.

FSD.INFO		JSR \$38(An)	
Call Parameters		Return Parameters	
Option 1:			
D1.B	Field number	D1.L	Field type.W, Length.W
D2		D2.L	Record no.W, Field no.W
Option 2:			
D1.B	0	D1.W	Current record number
D2		D2.L	Record count.W, Field count.W
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR . OR - Field number out of range			

There are two return options, depending upon what D1.B holds upon calling.
For Option 1, the returned field type is:

- 0 - String
- 1 - integer.W
- 2 - integer.L
- 3 - Floating Point

The field length for strings is the maximum length allowed, whether fixed or variable length strings are in use.

For Option 2, D2 . L holds the record count of all records in the database, in the high word, and the number of fields in the low word.

4.5.10 Order Database

Order a database.

FSD.ORDR		JSR \$3C(An)	
Call Parameters		Return Parameters	
D1.B	No. order parameters/0	D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Order parameter list	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR. OR - Field number out of range / too many parameters			

If D1.B is zero, then no ordering is done and the Order control block is released if in existence. This removes any order from the database.

No more than four parameters can be supplied.

The order parameter list pointed to by (A1) or (A6,A1.L) will be (D1.B * 2) words in size, up to a maximum of eight words. Each order parameter will have the format:

DC.W - Order Field number
 DC.W - Direction (1 ascending, -1 descending)

See Chapter 8 - **String Comparison Tables** for details on string comparing.

The Order control block is (re)allocated by this vector, unless A1 is zero. If an error occurs, then it is released before returning.

The Order sort block is allocated by this vector, and released prior to returning.

The current record buffer is returned, updated to the new, first ordered record.

4.5.11 Select/Unselect Records

Excludes or include any records in which the given expression is true.

FSD.SEL		JSR \$40(An)	
Call Parameters		Return Parameters	
D1		D1.W	New record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Select parameter list	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR . OR - Field number out of range			
ERR . BP - Bad parameter			

The parameter list pointed to by (A1) or (A6,A1.L), (applicable to FSD . SEL and FSD . FIND/FSD . CONT with FSDF . FSE) has a two byte header:

- Byte 0:
 - Bit 7 - 0 = use internal compare routine; 1 = use user supplied compare routine.
 - Bit 6 - 0 = Exclude selected records; 1 = include selected records.
 - Bits 5-0 = Not used.
- Byte 1: Number of parameters to follow. Zero = all records will be included or excluded.

The parameters to include or exclude records follow, beginning at byte 2. A maximum of four selection parameters can be supplied.

For FSD . SEL only:

Then for each parameter (4 maximum):

- \$0 .W Field number
- \$2 .B Condition mask, set bit:
 - \$0 for field is Greater-than is OK
 - \$1 for field is Equal-to is OK
 - \$2 for field is Less-than is OK
- \$3 .B Unused
- \$4 .L Offset of comparison data from (A1) or (A6,A1.L)
- \$8 .W Operation to link this parameter to next:
 - \$0 if no more parameters
 - \$4 to OR this result with next

- \$8 to AND this result with next
- \$C to XOR this result with next

If a user compare routine is to be used, as opposed to the internal routine, the word at \$0(A1) (or \$0(A6,A1.L)) will have bit 15 set. Bit 14 will be set to include records or clear to exclude records.

The long word at \$2(A1) (or \$2(A6,A1.L)) is the absolute address of the user routine. There will be no parameters as described above.

The user routine is called once for each record to be processed. FSD . COMP can be called from this routine.

4.5.11.1 User Compare Routine

The user supplied compare routine will take the following parameters.

Call Parameters		Return Parameters	
D2		D2.L	Undefined
D3		D3.L	Undefined
D4		D4.L	Undefined
D5.B	-ve if TRAP #4	D5.B	Result (bit 2 set if true)
D7 not saved/modified by the Database routines in any way			
A0.L	Channel ID	A0.L	Undefined
A1.L	Record address	A1.L	Undefined
A2.L	Routine Address	A2.L	Undefined
A3.L	Database Control Block	A3.L	Undefined
A4.L	Parameter list address	A4.L	Undefined
A6 not saved/modified by the Database routines in any way			

There will be an immediate exit on error.

The record is found (FSD . FIND/FSD . CONT) if D5 bit 2 is set (ie: the expression is true), similarly the record is selected (INCLUDE - FSD . SEL) or deselected (EXCLUDE - FSD . SEL) if D5 bit 2 is set.

FSD . SEL only:

The current record buffer is returned holding the first selected record. Note that the selection is superimposed upon any order, and calling the order vector will reset the selection.

The order control block is allocated if not already in existence, as this keeps the information about which records are selected (included) and which are deselected (excluded).

See Chapter 8 - [String Comparison Tables](#) for details on string comparing.

4.5.12 Reset Selection

Resets the record selection.

FSD.RES		JSR \$44(An)	
Call Parameters		Return Parameters	
D1		D1.W	New record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			

If the order control block was allocated by `FSD . SEL`, then `FSD . RES` releases the OCB, which effectively selects all records. However, if the OCB has been allocated by `FSD . ORDER`, the `FSD . RES` keeps the block, and selects all records.

`FSD . SEL` allocates the order control block if necessary but does not release it, so this vector is provided to do so.

If an order is imposed, then all this vector does is select every record. However, it won't release any memory, therefore keeping the order intact.

If an order is imposed, `FSD . ORDER` can be used to release the memory and remove the order.

The current record buffer is returned holding the first record.

4.5.13 Find Record

Find a record.

FSD.FIND FSD.CONT		JSR \$48(An) Find from beginning. JSR \$4C(An) Find from current position.	
Call Parameters		Return Parameters	
D1.W	Find type	D1.W	Record pointer or -1
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Find parameter list	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.NI - Invalid find type			
ERR.BP - Bad parameter			
ERR.OR - Field number / parameter out of range			

There are three find types as described in the following:

4.5.13.1 Find by Select

FSD.FSE with D1.W=\$0000 is Find by select type parameters. See FSD.SEL on Page 29 for a definition of the parameter list.

4.5.13.2 Find by INSTR/Numeric comparison

FSD.FIN with D1.W=\$0001 is Find by instr/number compare. The find parameter list pointed to by (A1) or (A6,A1.L) is:

- \$0(A1).W Type of parameter:
 - 0 = String
 - 1 = Word integer
 - 2 = Long integer
 - 3 = Floating point
- \$2(A1) Parameter.

All fields of the given type are searched; numeric fields by a straight comparison; string fields by an INSTR compare—if the parameter is "CAT" and the string field holds "caT" then a match will be found as the comparison is case independent.

Note

String parameters must include a length word.

4.5.13.3 Find by FSD . ORDR fields

FSD F . FOR where D1.W=\$0002 is Find by FSD . ORDR defined fields. The parameter list pointed to by (A6) or (A6,A1.L) is:

- \$0 .L Offset of parameter 1 from (A1) or (A6,A1.L)
- \$4 .L Offset of parameter 2 from (A1) or (A6,A1.L)
- \$8 .L Offset of parameter 3 from (A1) or (A6,A1.L)
- \$C .L Offset of parameter 4 from (A1) or (A6,A1.L)

Only as many fields as have been ordered are used, and parameter 1 is compared to order field 1, etc. Parameters can be supplied as null by setting their offsets to 0.

-1 is returned in D1.W if no record is found.

All finds are subject to ordering and selections.

See Chapter 8 - [String Comparison Tables](#) for details on string comparing.

4.5.14 Ordering Information

Return information about how a database is ordered.

FSD.OINF		JSR \$50(An)	
Call Parameters		Return Parameters	
Option 1:			
D1.B	0	D1.L	Parameter total or -1
D2		D2.L	Preserved
Option 2:			
D1.B	Parameter number	D1.L	Field no.W, Direction.W
D2		D2.L	Type.W
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR. OR - Parameter number out of range			

If D1.B=0 upon calling, then -1 is returned in D1 if no order is imposed, else the number of order parameters is returned. If D1.B<>0 then if no order is imposed, ERR.OR is returned.

The Type word is 0 for string, 1 for integer.W, 2 for integer.L, and 3 for Floating Point.

4.5.15 Close Database

Closes a database.

FSD.CLOS		JSR \$54(An)	
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Channel ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			

This vector only releases memory that has been attached to the database, and does not close the channel.

4.5.16 Import Database

Import a database from an Archive/Abacus type export file.

FSD.IMPT		JSR \$58(An)	
Call Parameters		Return Parameters	
D1		D1.W	Record count
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Channel ID	A0.L	Database ID
A1.L	Import channel ID	A1.L	Preserved
A2.L	Parameter list / 0	A2.L	Preserved
A3.L	MEMORY ALLOC	A3.L	Preserved
A4.L	Extra information buff	A4.L	Preserved
Error return:			
ERR . BO - Internal buffer full (Import field too long)			
ERR . BP - Field list wrong			

The parameter list pointed to by (A2) or (A6,A2.L):

- 0(A2) .W No. of parameters / 0
- 2(A2) .W 1st parameter
- 4(A2) .W 2nd parameter
- etc....

There is a one to one correspondence between parameters and fields, ie: parameter 1 -> field 1 parameter 2 -> field 2 etc.. There can be fewer parameters supplied than field names, in which case, defaults will be used where necessary. Extra parameters are ignored.

The first line of the import file is a list of field names, each of which give a clue as to how the field will be defined.

If the field name ends in "\$", then the field will be a string, the parameter will set the maximum string length (see FSD . CRT); the default being -128 (128-byte variable storage).

If the field name ends in "%" or "@", then the field will be word (type 1) or long word (type 2) types respectively; the parameter is ignored.

If the field name does not end in "\$", "%", or "@", then the parameter can specify the type as 1, 2, or 3 (see FSD . CRT); the default is floating point (type 3).

For example:

```
"Fred", "Jim$", "Edna%", "Tom"<CR><LF>
```

Without any parameters being supplied:

- Fred & Tom will end up as 6-byte floating point numbers.
- Jim\$ as a 128-byte maximum string.
- Edna% as a 2-byte integer.

However, if the following parameter block is supplied:

```
4; 2, 40, 3, 2
```

Then:

- Fred will be a 4 byte long integer.
- Jim\$ will be a 40 byte maximum fixed storage string.
- Edna% will be unaffected; a 2 byte integer.
- Tom will be a 4 byte long integer.

If A4 points to a standard QDOS string, then this is put into the extra information part of the file (see FSD.XTRS).

Otherwise if A4=0 then the first line of the import file is transferred as the extra information.

The import vector creates a temporary job to do the importing as some ROM vectors use A6-relative addressing; the calling job is suspended whilst this is in operation. See TRAP #1 - MT.ACTIV.

Large amounts of memory can be required for importing if one or more string fields have variable size. Initially the requirement is 9,168 bytes, but this only serves the first 2,048 records; an extra 4,128 bytes are required for every extra 2,048 records, to a maximum of 32,768 records.

See Section 4.4 - **Memory Allocation**, on Page 13, for an explanation of MEMORY ALLOC.

The current record buffer is returned holding the first record.

If an error occurs all allocated memory is released.

4.5.17 Export Database

Export a database to an Archive/Abacus type export file.

FSD.EXPT		JSR \$5C(An)	
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Export channel ID	A1.L	Preserved
A2.L	Parameter list	A2.L	Preserved
Error return:			
ERR .BO - Buffer overflow			

The parameter list pointed to by (A1) or (A6,A1.L):

- 0.W What to do with 1st export line:
 - 0 - output CR LF.
 - -ve - send appropriate names as stored in the name list (FSD .NMLD is issued as required).
 - +ve - send the following names, each in quotes, separated by commas, ending in CR LF.

If any names, then for each name:

- n+0.W Length
- n+2.B Characters of name (excluding quotes and LFs)

At the end of the names, If any have been supplied, should be a word of \$0000.
Followed by:

- f+0.W Number of fields to send (0 = send all)
- f+2.W 1st field to be sent
- f+4.W 2nd field to be sent, etc.
- Finally: \$0000.W

Note

A field can be sent once, more than once, or not at all.

The FSD .NMLD name handling vector is issued to load the name list if not in existence. If this happens, then the name list is removed when finished with.

The export vector creates a temporary job to look after the exporting due to the fact that some vectors in the main ROM use addressing relative to A6. Whilst this is in operation, the calling job is suspended by TRAP #1 - MT.ACTIV.

- Exporting is subject to both Selecting and Ordering of the database.
- The current record buffer is returned holding the last record.

See Chapter 7 - **Field Name Handling** for name handling information.

4.5.17.1 Export Entire Database

To export the entire database, with the field names as per the current database, the parameter list, pointed to by A2.L will be as per the following:

```
expParams
  dc.w  -1      ; Use database field names.
  dc.w   0      ; Export all fields.
  dc.w   0      ; End of parameter list
```

4.5.17.2 Export Two Columns

To export only two columns, with the field names as per the current database, the parameter list, pointed to by A2.L will be as per the following:

```
expParams
  dc.w  -1      ; Use database field names.
  dc.w   2      ; Export two fields.
  dc.w   1      ; Field one to be exported.
  dc.w   3      ; Field three to be exported.
  dc.w   0      ; End of parameter list
```

4.5.18 Update Extra Information

Overwrite extra information string.

FSD.XTRS		JSR \$60(An)	
Call Parameters		Return Parameters	
D1		D1.W	Length sent
D2.W	Buffer length	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Buffer address	A1.L	Preserved
A2		A2.L	Preserved
Error return:			

In each database is a string for storing application defined information. This is called the extra information string. The format should be:

"Field name 1","Name 2",...,"Name n"<CR><LF>User defined information

Field names should be stored in the string followed by <CR><LF> for FSD . EXP T, the name handling vectors, and applications. The name list is reloaded if in existence.

Room is made in the database for this information if needed, or the database is contracted and truncated if this is required. The extra information can be written into an export file if required.

See Chapter 7 - **Field Name Handling** for name handling information.

4.5.19 Fetch Extra Information

Fetch the extra information string.

FSD.XTRL		JSR \$64(An)	
Call Parameters		Return Parameters	
D1.B	Fetch type	D1.W	Length fetched / Info Length
D2.W	Buffer length / 0	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Buffer Address	A1.L	Preserved
A2		A2.L	Preserved
Error return:			

If D2=0, then A1 is ignored and D1.W is returned with the length of available information. Also the file pointer will be set to the beginning of the string (See TRAP #3 - FS.POSAB).

If D2>0, then (A1) or (A6,A1.L) point to the buffer, and D1.W holds the length fetched.

If D1=0 then IO.FSTRG is used to fetch the information, else IO.FLINE is used.

4.5.20 Set String Compare Tables

Set string compare table pointer.

FSD.SCPT		JSR \$68(An)	
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Pointer / 0 / -1	A1.L	Preserved
A2		A2.L	Preserved
Error return:			

A1 Meaning:

- -ve Case independent string compare.
- 0 Case dependent string compare. (DEFAULT)
- +ve User defined string compare.

The pointer is an absolute pointer to the user defined string compare table.

Affects:

- FSD.ORDER
- FSD.SEL
- FSD.FIND with FSDF.FSE
- FSD.CONT with FSDF.FSE
- FSD.FIND with FSDF.FOR
- FSD.CONT with FSDF.FOR
- FSD.COMP

See Chapter 8 - **String Comparison Tables** for details on string comparing.

4.5.21 Compare Values

Compare strings, integers, or floating point numbers.

FSD.COMP		JSR \$6C(An)	
Call Parameters		Return Parameters	
D0.W	Field type	D0.L	saved CCR
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Undefined
A0.L	Pointer to buffer 1	A0.L	Preserved
A1.L	Pointer to buffer 2	A1.L	Preserved
A3.L	Database ID	A3.L	Preserved

The field type can be:

- 0: Qdos type string
- 1: Word integer
- 2: Long integer
- 3: Floating point

Example subroutine:

```

JSR    FSD.COMP (A2)
MOVE   D0,CCR
BEQ.S  EQUAL
BGT.S  GREATER
LESSER
...
```

If it is to be used with the buffers in a *SuperBASIC* area (*Do not* use TRAP #4):

```

TRAP   #0
ADDA.L A6,A0
ADDA.L A6,A1
JSR    FSD.COMP (A2)
ANDI.W #$DFFF,SR
MOVE   D0,CCR
BEQ.S  EQUAL
BGT.S  GREATER
LESSER
...
```

This routine will only compare buffers of the same type, eg: string and string; or: floating point and floating point.

The Database ID is required to get the address of the current string compare table, and is not necessary for numeric compares.

See Chapter 8 - [String Comparison Tables](#) for details on string comparing.

4.5.22 Open Directory

Open a device's directory as a database. A channel must be opened before use, but as usual, it is not closed by any of the vectors.

FSD.DDIR		JSR \$70(An)	
Call Parameters		Return Parameters	
D1		D1.L	Record no.W, field no.W
A0.L	Channel ID	A0.L	Database ID
A2.L	MEMORY ALLOC	A2.L	Preserved
Error return:			
ERR.NO - Channel not open			

The channel supplied to this vector must be opened with `IO.DIR` (see the `IO.OPEN` trap, register D3).

`FSD.SOKY` should be called after calling this vector to set the order key size to at least 12.

See Section 4.4 - Memory Allocation, on Page 13, for an explanation of `MEMORY ALLOC`.

If an error occurs, then all allocated memory is released before returning. An extra information string is stored in the code, and can be accessed with `FSD.XTRS` and the name handling vectors.

See Chapter 9 - Directory Support Using Database Vectors for further information.

4.5.23 Load Code Section

Load the contents of the code section into the supplied buffer.

FSD.CDLD		JSR \$74(An)	
Call Parameters		Return Parameters	
D1		D1.W	Length fetched / available
D2.W	Buffer length / 0	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Buffer Address	A1.L	Preserved
A2		A2.L	Preserved
Error return:			

If D2 is zero, then A1 is ignored and D1.W is returned with the length of the code section. The file pointer will be set to the beginning of the code (See TRAP #3 - FS.POSAB).

If D2>0, then (A1) or (A6,A1.L) point to the buffer, and D1.W holds the length fetched.

See Chapter ?? - ?? for a description of how to use a code section.

4.5.24 Database Space

Makes or removes space in the database file.

FSD.UPF		JSR \$78(An) Make space	
FSD.DNF		JSR \$7C(An) Remove space	
Call Parameters		Return Parameters	
D1		D1.L	Undefined
D2		D2.L	Undefined
D3		D3.L	Undefined
D4.L	Memory alloc address	D4.L	Preserved
A0		A0.L	Channel ID
A1		A1.L	Undefined
A2		A2.L	Undefined
A3.L	Database Address (ID)	A3.L	Preserved
Error return:			

In both vectors REC.LEN(A3) holds the amount of space to be made/removed. In FSD.UPF, REC.FPTR(A3) points to the place where space is to be made. Whereas in FSD.DNF, REC.FPTR(A3) points to the end of the space to be removed.

Both vectors allocate a piece of memory to store parts of the file in as they are being moved about. The value in DB.MEMO(A3) should be put in D4.L before calling either vector.

The file is truncated after FSD.DNF is called.

Both vectors update DBH.LTAB(A3). However, if the space is before the location pointed to by DBH.RPT(A3) in the file, then this pointer should be adjusted afterwards, and relevant part of the header saved.

Do not use these vectors to remove/make space at the end of the file. Also, if the space is after the location pointed to by DBH.LTAB(A3), then these vectors *should not* be used.

If errors ERR.RO or ERR.DF occur in FSD.UPF, then the file and the database control block *should be* uncorrupted.

Use these vectors with care as the database file could well be corrupted, with extra bits added, or bits missing.

4.5.25 Save Code section

Set the contents of the code section in the database file, from the supplied buffer.

FSD.CDSV		JSR \$80(An)	
Call Parameters		Return Parameters	
D1		D1.W	Length stored
D2.W	Buffer length	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Buffer Address	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.OV - code will not fit in header block			

Room is made in the database for this code if needed, or the database is contracted and truncated if this is required.

See Chapter ?? - ?? for a description of how to use a code section.

4.5.26 Get Field Name Vectors

Get the address of the field name vectors.

(FSD.NAME	JSR \$84(An)
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0		A0.L	Preserved
A1		A1.L	Preserved
A2		A2.L	Vector address
Error return:			
ERR.NI - not implemented in this version			

This vector returns the address of the field name handling vectors. These use the names stored in the proper format in the extra information string, and allow fetching, setting, and searching for field names.

See Chapter 7 - **Field Name Handling** for further information.

4.5.27 Get Subfield Vectors

Get the address of the subfield vectors.

FSD.SUBF		JSR \$88(An)	
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0		A0.L	Preserved
A1		A1.L	Preserved
A2		A2.L	Vector address
Error return:			
ERR.NI - not implemented in this version			

This vector returns the address of the subfield handling vectors. These use the names stored in the proper format in the extra information string, and allow fetching, setting, and searching for field names.

See Section [5.8.13 - Get Subfield Contents](#) on Page [69](#) for further information on subfields.

4.5.28 Get Extra Vectors

This vector returns the address of the extra vectors. This is required to allow more vectors than the original vector table will hold.

FSD.XVEC		JSR \$90(An)	
Call Parameters		Return Parameters	
D1		D1.L	Preserved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0		A0.L	Preserved
A1		A1.L	Preserved
A2		A2.L	Vector address
Error return:			
ERR.NI - not implemented in this version			

See Chapter 7, Subsection 7.5.2 - [Extra Vectors](#) on Page 83 for further information.

Chapter 5

DBAS SuperBASIC Interface

5.1 Loading the *SuperBASIC* Interface

SuperBASIC (or a *MultiBasic* if you have Minerva) can be extended by:

```
b=respr(9*1024): lbytes flp1_DBAS_BIN,b: call b
```

or, if you have Toolkit 2:

```
lrespr flp1_DBAS_BIN
```

DATA_BIN must be loaded before DBAS_BIN can be used.

5.2 Symbols Used

The following symbols are used when documenting the parameters required by the various *SuperBASIC* procedures and functions provided by the DBAS_BIN file.

- C = Channel number
- C\$ Comparison string (a combination of "=", "<" and ">")
- D = Direction (+1=ascending, -1=descending)
- D\$ = Database file name
- E\$ = Export file name
- F = Field number (or a field name where appropriate)
- FD = Found flag (0=not found)
- FL = Field length (string only)
- FT = Field type
 - 0 = string
 - 1 = word integer
 - 2 = long word integer
 - 3 = floating point

- I\$ = Extra information
- N\$ = Export field name
- O\$ = Operation identifier ("AND"/"OR"/"XOR")
- R = Record number
- RC = Record count
- SF = Subfield number
- SO = Subfield offset
- V = Value
- () = Optional item
- ** = Repeatable item

5.3 General

If the `DATA_USE` procedure from Toolkit II is available, then the default device name will be added on the front of an invalid filename in an attempt to make a valid one.

Procedures with an `_O` ending their name (eg: `CREATE_O`) will overwrite an existing file automatically, whereas an equivalent procedure without the `_O` (eg: `CREATE`) will ask for permission before overwriting.

5.4 Memory Allocation

Certain procedures allocate or release blocks of memory for use by `DATA_BIN`. Eg: `IMPORT`, `EXCLUDE`. If an error occurs, any memory allocated by that particular procedure is released. Eg: `ORDER` will release memory allocated by itself, but not memory allocated by `CREATE`.

Memory is allocated from the Common Heap in the same way as for `ALCHP` (see Toolkit II). With this method, *SuoperBASIC* keeps a list of the pieces of memory allocated, and a `CLCHP` procedure is provided in Toolkit II to remove all the pieces. The result being that `CLCHP` will remove system database memory as well.

Note

Toolkit II is not required for use of this package; but without it, if a `CLOSE` is performed on a database, rather than a `CLOSE_DATA`, the pieces of memory could be lost "permanently". Therefore a `CLCHP` procedure is provided in `CLCHP_BIN`. To load and use it:

```
a=RESPR(70): LBYTES FLP2_CLCHP_BIN,a: CALL a
CLCHP
```

On *Minerva* each Basic has its own independent list, and so a `CLCHP` issued in one Basic will not affect another's memory.

5.4.1 Database Blocks

The various database blocks are:

Database Control Block	Holds most of the information to allow the database system to work.
Current Record Buffer	Holds the current record data.
Record Length Buffer	Holds a section of the in-file record length table.
Order Control Block	Holds store of record offsets, which define the order of the database.
Order Sort Block	Temporary block, only allocated during ORDER, used to store key fields.
Name Table	Holds offsets and lengths of names in the extra name list.
Name List	Holds the first part of the extra information string (up to the first LF) which contains the field names.

5.5 Field Names

The use of field names in accessing fields is available, but not as it is in Archive; fields have to be accessed using field numbers. Eg:

```
ORDER #3;1,1
ORDER #3;"FRED",1
```

However:

```
LET FRED=1
ORDER #3;FRED,1
```

can be used instead.

However, to allow the use of field names, the Extra Information string (see SEXTRA) must be set to the format:

```
"Field Name 1", "Field 2", ..., "Field k"<CR><LF>User defined string
```

The field names can be up to 255 characters long, but 20 will be all that is displayed by the ALTER_BIN program.

5.6 Array Parameters

In some procedures (eg: CREATE) lists of parameters can be replaced with an array of the specified type.

Arrays can be set to 1 element only, eg: DIM F%(0); or one set of entries, eg: DIM F%(0,n) or DIM N\$(0,m).

5.6.1 Procedures Summary

The following procedures are provided.

ADD_FIELD #C;F,FT,FL	Add a new field
APPEND #C	Add a new record
COPY_DATA(_O) #C TO D\$	Copy a database
CREATE(_O) #C;D\$,FT,FL *,FT,FL*	Create a database
CLOSE_DATA #C	Close a database
EXCLUDE #C;F,C\$,V *,O\$;F,C\$,V*	Deselect records
EXCLUDE #C	Deselect all records
EXPORT(_O) #C TO E\$ *;N\$* *F*	Export a database
FIND(C) #C; (FT,)V	Find by INSTR
IMPORT(_O) E\$ TO #C;D\$ *,T*	Import a database
INCLUDE #C;F,C\$,V *,O\$;F,C\$,V*	Select records
INCLUDE #C	Select all records
LOAD_NAMES #C(;V)	Reload name list
LOCATE(C) #C;V1(,V2(,V3(,V4)))	Find by ORDER parameters
OPEN_DATA #C;D\$	Open database - modifyable
OPEN_DDIR #C;D\$	Open database - directory
OPEN_DIN #C;D\$	Open database - read only
ORDER #C	Unorder a database
ORDER #C;F,D(;F,D(;F,D(;F,D)))	Order a database
REMOVE #C	Delete a record
REMOVE_FIELD #C;F	Remove a field
REMOVE_NAMES #C	Remove field names
RESET #C	Reset record selection
RPOSAB #C;R	Goto absolute position
RPOSRE #C;R	Goto relative position
SAVE_NAMES #C	Save field names
SCPTR #C(;Pointer)	Set compare table
SEARCH(C) #C;F,C\$,V *,O\$;F,C\$,V*	Find by INCLUDE parameters
SET #C;F,V	Set a field
SEXTRA #C;I\$	Set extra information
SUBF_ADD #C;F,SF,N	Add subfields
SUBF_REM #C;F,SF1 TO SF2	Remove subfields

SUBF_SET #C;F,SF,V\$	Set a subfield by number
SUBF_SETO #C;F,SO,V\$	Set a subfield by offset
SORDKEY #C;N	Set order key length
STNAME #C;F,V\$	Set a field name
UPDATE #C	Update a record

5.6.2 Functions Summary

COUNT (#C)	Get record count
DBADDR (#C)	Get the control address
FETCH (#C;F)	Get field contents
FEXTRA (#C)	Get extra information
FLEN (#C;F)	Get field length
FLNAME (#C;F)	Get a field name
FLNFIND (#C;V\$)	Get number of field name
FLNUM (#C)	Get field count
FLTYP (#C;F)	Get field type
FOUND (#C)	Get found flag
RECNUM (#C)	Get current record number
SUBF_FETCH (#C;F,SF[,SO%])	Get subfield
SUBF_FCURRE (#C;F,SO[,SO%])	Get subfield by offset
SUBF_FNEXT (#C;F,SO[,SO%])	Get next subfield
SUBF_NUM (#C;F[,SO])	Get subfields count/number

Also, all procedures have function equivalents which return the error code as a float. Their names are FDB_ with the procedure name appended, eg:

```
error=FDB_CREATE(...)
error=FDB_IMPORT_O(...)
error=FDB_CLOSE_DATA(...)
```

5.7 Procedures

5.7.1 Add a Field

```
ADD_FIELD #C;F,FT,FL
```

The database attempts to add a field to a database. The field is inserted in front of field F, or appended if F is larger than the field count. FT and FL specify the field type as for create. STNAME should be called after this to set the field name.

5.7.2 Add a Record

```
APPEND #C
```

An ordered database is always reordered after this instruction has been issued. APPEND uses the contents of the *current record buffer*, not the contents of variables.

5.7.3 Copy Database to New Database

```
COPY_DATA #C TO D$
COPY_DATA_O #C TO D$
```

Copy a database to the given file. The selected records are copied in the order imposed. The file is in effect created as a new database. Its name list and extra information are as stored in the original database file.

This will also work on a directory database.

5.7.4 Create Database

```
CREATE #C;D$,FT,FL *,FT,FL*
CREATE_O #C;D$,FT,FL *,FT,FL*
CREATE #C;D$,F%
CREATE_O #C;D$,F%
```

Each FT, FL describes one field, FT being the type, and FL being the length in the case of a string. FL does not have to take into account the storage of the string length, as the machine code vector adds this in. If FL is negative for a string, then the string is variable length, and `abs(FL)` is the maximum string length.

F% is an integer array, created by `DIM F%(n,1)`. Note: $0 \leq n \leq 254$. Array index `F%(m,0)` is the FT while index `F%(m,1)` is the FL. Note that m goes from 0 to n, and so n+1 entries are created, each of which corresponds to a field.

There is a maximum of 255 fields for each record, a maximum length of 32767 bytes for each record, and a maximum number of 32767 records.

The database control block and current record buffer are allocated at this point.

This procedure also creates and loads a blank field name list.

5.7.5 Close Database

```
CLOSE_DATA #C
```

This gets rid of all allocated memory associated with the database, and then closes the channel. Note that `CLOSE #C` is not required as this is carried out by `CLOSE_DATA`.

5.7.6 Exclude Records

```
EXCLUDE #C;F,C$,V *,O$;F,C$,V*
```

```
EXCLUDE #C
```

Deselect any record in which the expression is true. See [5.7.10](#) - `INCLUDE` on Page [59](#) for a description.

5.7.7 Export Database

```
EXPORT #C TO E$ *;N$* *F*
```

```
EXPORT_O #C TO E$ *;N$* *F*
```

Note

`*;N$*` can be replaced by array: `DIM N$(n,m)`.

`*F*` can be replaced by array: `DIM F%(k)` or `DIM F$(k,1)`

Note that a `\` (backslash) must be used as the separator directly before the first `F`, `F%`, or `F$`, and nowhere prior to that; eg:

```
EXPORT #3 TO FLP2_A;"f1","f2"\1,2
```

is correct, whereas:

```
EXPORT #3 TO FLP2_A\"f1","f2"\d%
```

is not.

This procedure will export a database to an Archive/Abacus type export file. All N\$'s (or the contents of array N\$) are put out on the first line of the export file, enclosed in quotes and separated by commas.

If no N\$'s are supplied, the appropriate names from the name list (see `LOAD_NAMES`) are output as line 1. (If no name list is loaded, it is then loaded temporarily, and removed when finished with).

All lines in the export file end in CR, LF. A CTRL-Z is put at the end of the file.

If no F's are supplied, then all fields are output once in the order that they were defined. Otherwise the F's (or array F%) say what fields are exported, in what order, and how many times.

Records are exported in order if an `ORDER` has been imposed, and also only those that have not been deselected.

5.7.8 Find Records

```
FIND #C; (FT, ) V
FINDC #C; (FT, ) V
```

Find a record with the given value in any field of a given type. FT need not be given if string fields are to be searched. FT must be given if numeric fields are to be searched.

- 0 specifies a string search (default)
- 1 specifies a word integer search
- 2 specifies a long word integer search
- 3 specifies a floating point search.

To be found, numeric fields must be an *exact* match. String fields must contain the given string somewhere within, and the match is case independent, (i.e. string fields are tested using `INSTR`).

`FIND` searches from record 0, whereas `FINDC` searches on from the current record pointer. Only records which are selected are searched, and records are searched in order if an `ORDER` has been imposed.

The result is returned by the function `FOUND (#C)`.

5.7.9 Import Database

```
IMPORT E$ TO #C; D$ *, T*
IMPORT_O E$ TO #C; D$ *, T*
IMPORT E$ TO #C; D$, T%
IMPORT_O E$ TO #C; D$, T%
```

This procedure will import an Archive/Abacus export file into a new database. These procedures, `IMPORT` and `EXPORT`, temporarily create a special job to do the work. This job is owned by the Basic that issued the procedure.

`#C` is attached to `D$` as for `CREATE` and `CREATE_O`, though `IMPORT` decides what the fields are.

`T`'s are parameters that supply overriding additional information to the special job about the fields it is going to create. Each `T` applies to one field only; the first `T` applies to the first field, and the second `T` applies to the second field, etc.

There do not have to be as many `T`'s as fields, and any extra `T`'s are ignored. However, `T`'s cannot be skipped, eg: 1,3,,4 is not allowed.

`T%` is an integer array, created by: `DIM T%(n)` [note `n` can be 0]. Where `T%(m)=T`. Note: `0 <= m <= n`. This means `n+1` entries are created.

Field Name Ending	Field Type	T	Default
\$	String	length 128	Variable length
%	Int.W ignored		
@	Int.L ignored		
none	numeric	type 1-3	Floating point, 3

Large amounts of memory can be required for this facility if one or more variable length string fields are used. The initial requirement is 9,168 bytes for the first 2,048 records, followed by an extra 4,198 bytes for every further 2,048 records thereafter, until the maximum of 32,768 records is reached.

This procedure puts the first line of the export file into the extra information string, and loads it as a name list.

5.7.10 Select Records

```
INCLUDE #C;F,C$,V *;O$;F,C$,V*
INCLUDE #C
```

Select any record in which the given expression is true. If no extra parameters are supplied, then the expression is always true, so all records are de(Or selected if using `EXCLUDE`).

- `F` = Field number
- `C$` = Compare type (combination of `=`, `<`, `>`) in quotes.
- `V` = Value to compare
- `O$` = Linkage operator (`AND/OR/XOR`) in quotes.

Eg:

```
INCLUDE #3;1,"=",14; "OR";2,"<","DOG"; "AND";3,">",14
      ^^^^^^  ^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^
      term 1    term 2                term 3
```

is the same as the unimplemented:

```
INCLUDE #3; (FETCH(#3;1)=14 OR FETCH(#3;2)<"DOG") AND FETCH(#3;3)>14)
      ^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  ^^^^^^^^^^^^^^^^^^^^^^^^^
      term 1            term 2                        term 3
```

Note: The order in which terms are supplied is significant, rather than the normal priority system (which is AND>OR and OR=XOR). A maximum of 4 terms are allowed.

EXCLUDE/INCLUDEs can be overlaid upon each other, eg:

```
REMark: This doesn't seem to be used. (ND)
LET Household=1: LET pets=2: LET type=3

REMark Include all records.
EXCLUDE #3

REMark Include only households with pets.
INCLUDE #3; pets, "=", "Yes"

REMark But exclude any cats and dogs.
EXCLUDE #3;type, "=", "Cat"; "OR"; type, "=", "Dog"
```

The result of this is that all households with no pets, or those that have cats or dogs are deselected; leaving selected only those households with pets other than cats or dogs.

This procedure allocates the Order Control Block, as this holds selection information.

5.7.11 Load Field Names

```
LOAD_NAMES #C(;V)
```

Load the field names from the database, and allow access by other procedures and functions. It is automatically issued by CREATES, OPENS, and IMPORTS.

If V is given, this is the minimum average name storage length (the names are stored contiguously, so one name can be longer, and another shorter). If more storage is required to load in successfully, more is obtained. The default is 20.

SEXTRA reissues this command if necessary.

5.7.12 Search Ordered Records

```
LOCATE #C;V1(,V2(,V3(,V4)))
LOCATEC #C;V1(,V2(,V3(,V4)))
```

Search for records using ORDER fields to compare with. The first record *after* the record where the supplied values will fit is found if an exact match is not. LOCATE begins from the start, LOCATEC continues from the next record.

ORDER—see Page 62—must have been issued previously or no search will occur. Any order fields for which no values have been supplied are not used. Any extra values supplied to the procedure will cause an error.

The following examples are based on this ordering:

```
ORDER #3;1,1; 4,-1; 3,1
```

```
LOCATE #3;4,"DOG",ln 2
```

Will look for the first record equal to or after the given set of values on fields 1, 4, and 3.

```
LOCATE #3;2,"CAT"
```

Will look for the first record equal to or after the given set of values on fields 1 and 4 only.

```
LOCATEC #3;45,"CAT",23,"Tabby"
```

Will produce an error, because "Tabby" does not have an order field assigned

```
LOCATE #3;32,,31
```

Will generate an error because values cannot be skipped.

Only records which are selected are searched.

5.7.13 Open Database

```
OPEN_DATA #C;D$
OPEN_DIN #C;D$
```

Open an existing database. OPEN_DATA opens in read/write mode and allows modification to the database; OPEN_DIN opens the database in read only mode.

These two procedures allocate memory as in the same way that CREATE does. The name list is loaded, or a blank one created.

5.7.14 Open Device Directory

```
OPEN_DDIR #C;D$
```

This procedure will open a directory device's directory as a database. This allows a disk/microdrive directory entries to be accessed using the database routines.

A name list is held in the DATA_BIN file, it is "loaded" as the name list. The extra information string is the name list followed by the medium name of the device being opened.

The order key size (SORDKEY) should be increased to, say, 12 before using order.

See Chapter 9 - [Directory Support Using Database Vectors](#) on Page 91 for further details

5.7.15 Order Database

```
ORDER #C
ORDER #C;F,D(;F,D(;F,D(;F,D)))
```

The first example will unorder a database, and release Order control block, NB: this also does a RESET.

The second example will order a database.

Up to four fields can be specified for ordering upon, and the first is the most significant, the last least. D is +ve for ascending order and -ve for descending. Any field can be used in order.

This procedure allocates the Order control block and Order sort block. The Order sort block is released at the end.

Note: This also does a RESET.

5.7.16 Delete Record

```
REMOVE #C
```

An ordered database is always reordered after this instruction is been issued.

REMOVE operates on the current record, and therefore cannot affect deselected records.

5.7.17 Remove Field

```
REMOVE_FIELD #C;F
```

Note that if one of the fields is an order key, the database will subsequently be unordered. The last remaining field on a record cannot be removed.

5.7.18 Remove Filed Name Memory

REMOVE_NAMES #C

This procedure will remove the field name handling memory.

5.7.19 Select All Records

RESET #C

Select all records in a database. Releases the Order Control Block if no order has been issued. This procedure resets the database back to state as that upon opening or creating it.

5.7.20 Position Record Pointer

RPOSAB #C;R

RPOSRE #C;R

Position record pointer absolute, or relative to the current record.

These procedures will overwrite what is in the internal record buffer with data from the new current record after repositioning.

ORDER, INCLUDE and EXCLUDE affect which records can become the current record, and in what order they occur. Deselected records are totally ignored, in other words, Record 0 is the first *selected* record; Record 1 is the second *selected* record and so on.

If the given record number is greater or equal to the number of selected records in the file, then COUNT () - 1 will be used instead.

5.7.21 Save Name List

SAVE_NAMES #C

Saves the name list out into the extra information section, overwriting the previous name list. Not implemented on directories (OPEN_DD IR).

5.7.22 String Comparison Order

SCPTR #C(;Pointer)

Sets the pointer to a "translate" table used for string comparing in: EXCLUDE; INCLUDE; LOCATE; LOCATEC; ORDER; SEARCH; SEARCHC. Pointer can be one of:

- Omitted - case dependent (same as 0)
- -ve - case independent
- 0 - case dependent (DEFAULT)
- +ve - pointer is an absolute address to user defined table

Each database channel has its own pointer.

See Chapter 8 - **String Comparison Tables** on Page 87.

5.7.23 Search Records

```
SEARCH #C;F,C$,V *,O$;F,C$,V*
SEARCHC #C;F,C$,V *,O$;F,C$,V*
```

Searches the database for a record in which the given expression is true.

- F = Field number
- C\$ = Compare type (combination of =,<,>)
- V = Value to compare
- O\$ = Linkage operator (AND/OR/XOR)

Examples:

```
SEARCH #3;1,"=", 14, "and"; 2, "<", "cat"
^^^^^^^^ ^^^^^^^^^^^^^^^^^^^^^^^^^
term 1    term 2
```

is equivalent to:

```
SEARCH #3; FETCH(#3;1)=14 AND FETCH(#3;2)<"cat"
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
term 1          term 2
```

SEARCH searches from the beginning, SEARCHC starts from the next record, until a record is found in which the given expression is true. The records are searched in order, if one is imposed, and only records which are selected are searched.

At present, only 4 terms are allowed.

5.7.24 Set Field Value

```
SET #C;F,V
```

This sets the contents of a field stored in the current record buffer. It is not written out until a subsequent APPEND or UPDATE is issued.

5.7.25 Save Extra Information

SEXTRA #C;I\$

Store extra information, erasing previous information. I\$ must be of the format:

"field name 1", "field name 2", .. "field name n" <CR> <LF> User defined text

Eg:

' "Book_number", "Author", "Book_title" ' &CHR\$(13) &CHR\$(10) &'My Books Database'

Note

Each field will have its name in double quotes, single quotes are not allowed. The extra information, after <CR><LF>, can be in single quotes. The entire set of field names, wrapped in their double quotes, must then be wrapped in a pair of single quotes.

Remembering this will save you wondering why your database doesn't appear to have any field names!

There must be one field name per field.

The extra information prior to the 1st <CR><LF> is used for storing field names. Only one string is allowed per database. The <CR><LF> should be included to allow EXPORT to produce a valid export-type file.

This procedure automatically reissues LOAD_NAMES if needed.

This will work, even if records are present in the file. If necessary, it creates/removes space in/from the file.

This procedure is not implemented for a directory (OPEN_DD IR).

5.7.26 Order Key Size

SORDKEY #C;N

Sets the order key length. A longer key allows a more thorough comparison, but increases the time taken to perform the order.

5.7.27 Rename Field

STNAME #C;F, V\$

This command does not save the new field name on the file, SAVE_NAMES must be issued to do that.

5.7.28 Add Subfields

SUBF_ADD #C;F,SF,N

Add a number (N) of subfield markers in front of subfield SF, in field F. If SF is higher than the number of subfields, the subfield markers are appended to the field.

5.7.29 Remove Subfields

SUBF_REM #C;F,SF1 TO SF2

Remove subfields SF1 to SF2 from field F. The associated field markers are also removed. Note that a marker belongs to the preceding field. To remove a single subfield, simply ensure that SF1 equals SF2.

5.7.30 Replace Subfield Content

SUBF_SET #C;F,SF,V\$

SUBF_SETO #C;F,SO,V\$

Replace a subfield's contents. Refer to the subfield by subfield number (SF) or offset within the field (SO).

5.7.31 Update Record

UPDATE #C

Update the current record. An ordered database is always reordered after this instruction is issued.

UPDATE uses the contents of the current record buffer, not the contents of any variables. UPDATE operates only on the current record, so cannot affect deselected records.

5.8 Functions

The following functions are provided.

5.8.1 Count Selected Records

COUNT (#C)

Return the number of selected records, which will be the number of record in the file if all records are selected.

Remember that the record number of the last record is $\text{COUNT}(\#3) - 1$ as the records numbers start at zero!

5.8.2 Database Control Block Address

DBADDR (#C)

Get the address of the database control block.

5.8.3 Fetch Field

FETCH (#C; F)

Return the value of the field from the current record block.

5.8.4 Fetch Extra Information String

FEXTRA (#C)

Return the extra information string (set by SEXTRA).

5.8.5 Field Length

FLEN (#C; F)

Return the maximum length of field F, excluding any string length storage space.
Note: This is always positive, even for variable length fields.

5.8.6 Field Name

FLNAME (#C; F)

Return the field name for field F. This is the only case where F cannot be a field name.

5.8.7 Field Number

FLNFIND (#C; V\$)

Return the field number corresponding to field name V\$. 0 is returned if no match found.

5.8.8 Field Count

FLNUM (#C)

Return the number of fields.

5.8.9 Field Type

FLTYP (#C; F)

Return the type of field F.

- 0 string
- 1 word integer
- 2 long word integer
- 3 floating point

5.8.10 Error Functions

FDB_XXXX (. . .)

Return the error code produced. These function perform same task as procedure named by xxxx. Eg: FDB_OPEN_DDIR (#C; D\$) .

5.8.11 Did Search Succeed

FOUND (#C)

Return 1 if found, 0 if not.

Affected by FIND, FINDC, LOCATE, LOCATEC, SEARCH, and SEARCHC.

5.8.12 Return Current Record Number

RECNUM (#C)

Returns the record number of the current record.

5.8.13 Get Subfield Contents

- SUBF_FETCH (#C; F, SF [, SO%])
- SUBF_FCURRE (#C; F, SO [, SO%])
- SUBF_FNEXT (#C; F, SO [, SO%])

A null string is returned if the given subfield number (SF) is not found in field F. Alternatively, a subfield offset can be given, and either the subfield starting there (SUBF_FCURRE) or the subfield following the next marker after there.

If SO% is given (an integer variable), then the variable will be set to the offset of the subfield within the field.

5.8.14 Subfield Numbers

SUBF_NUM (#C; F [, SO])

If no offset (SO) is given, count the subfields in field F. If the field is numeric, -1 will be returned.

If the offset is given, then return the number of the subfield containing the given offset. If the offset is beyond the end, -1 will be returned.

Chapter 6

C68 Library

LibDBAS is the Database library for the C68 compilation system. To use this library DATA_BIN or DROM_BIN will have to be CALLED prior to use.

Note

Any errors are put into `_oserr` and `_dberr`.

The following is a list of the structures, unions and functions contained in the C68 `libdbas_a` Database library.

6.1 Structures and Unions

6.1.1 Field List Structure

Field list structure for `fsd_excl`, `fsd_incl`, `fsd_srch` and `fsd_srch_c` compare routine.

Warning

These routines now supply the `FLD_LIST` array with [1] as field 1 instead of [0].

```
struct fld_list {  
    short fld_ptr; /* Offset of field from record start */  
    char fld_dyn; /* Bit 7 set if dynamic field */  
    char fld_typ; /* Field type:  
                  0 = String  
                  1 = Word integer
```

```

                2 = Long integer
                3 = Floating point. */
    short fld_len; /* Storage length of field */
};

```

6.1.2 Parameter Definitions

Parameter definition usable for some routines:

```

union fsd_parameter {
    struct QLSTR type0;
    short int type1;
    long int type2;
    struct QLFLOAT type3;
};

```

Also:

```

typedef union fsd_parameter fsd___p;

```

6.2 Library Functions

6.2.1 Add Field

```

short fsd_addf(
    long *dbid,
    int field,
    int type,
    int length)

```

Add a field of the given type & length in front of the given field. Note the database ID is changed by this.

Returns +ve new field count, -1 on error, -2 on fatal error. See FSD .ADDF.

6.2.2 Add Record

```

short fsd_appn(long dbid)

```

Add a record.

Returns +ve selected record quantity, -1 on error. See FSD .APPN.

6.2.3 Close Database

```
long fsd_close(long dbid)
```

Closes the database and release all the memory associated with a database.

Note

Does not close the associated channel.

Returns `_oserr`.

See `FSD.CLOS`.

```
char fsd_comp(long dbid, int type, void *par1, void *par2)
```

Compare two parameters using the internal compare routine, and therefore the string compare table. (See `fsd_scpt`). type = 0: struct QLSTR 1: short int 2: long int 3: struct QLFLOAT Return = -ve: par2 < par1 0 : par2 = par1 +ve: par2 > par1 `_oserr` unaffected. See `DATAREF_doc`, `FSD.COMP`.

Create a database in the FILE channel specified. field list: type 1 (0=Str 1=short 2=long 3=QLfloat) size 1 type 2 size 2 ... Returns +ve Database ID (dbid), -ve `_oserr` on error See `DATAREF_doc`, `FSD.CRT`.

Copy a database. The selected records are copied, in order. Returns +ve no. records copied, -1 on error See `XVECTORS_doc`, `FSD.COPY`.

Return the current record number. `_oserr` and `_dberr` unaffected. See `DATAREF_doc`, `FSD.INFO`.

Confirm database and get the database control block address. Returns non-zero address, 0 on error See `XVECTORS_doc`, `FSD.CBAD`.

Delete the current record. Returns +ve selected record quantity, -1 on error See `DATAREF_doc`, `FSD.DEL`.

Exclude records using a user-written C function to test a record. Sbrr is the address of a routine, or it can be 0 to act up on all records. Returns +ve selected record quantity, -1 on error. See DATAREF_doc, FSD.SEL.

User written C function: char sbrr(long dbid, void *record, struct fld_list list[], void *application) Set _oserr with any error. Note 1: the name of the routine can be anything. Note 2: the application parameter can be any pointer, and is passed to the user-defined routine. Note 3: list[1] is the first field, list[0] being irrelevant. If the char is returned FALSE then that record is passed over, else it is made unavailable.

Export a database to file exptid. param points to the parameter block, or can be 0. Returns _oserr. See DATAREF_doc, FSD.EXPT.

Find any field of the given type (first word of param) that is equal to the parameter supplied (remainder of param); or in the case of strings, contains the param somewhere within (ie: tested using INSTR). Returns +ve record position, -1 if not found, -2 on error.

```
struct find_param {
short type;
union fsd_parameter par;
} param;
```

(Not defined in header) See DATAREF_doc, FSD.FIND/FSD.CONT.

Return +ve the maximum field length, -1 on error. See DATAREF_doc, FSD.INFO.

Get a field from the current record buffer and place into fbuff. The name has a NULL char on the end, making it a C string. fsd_load_names must be issued first. If fbuflen = 0, the length available is returned. Returns +ve length moved/available, -1 on error. See NAMES_doc, FSD.NMGT.

Get the field number that corresponds to the name in the given C string. fsd_load_names must be issued first. Returns +ve field number, 0 not found, -1 error. See NAMES_doc, FSD.NMFN.

Return the quantity of fields. _oserr and _dberr unaffected. See DATAREF_doc, FSD.INFO.

Return +ve the field type, -1 on error. See DATAREF_doc, FSD.INFO. _____

Get a field from the current record buffer and place into fbuff. If fbuflen = 0, Returns the length available. Returns +ve length moved, -1 on error See DATAREF_doc, FSD.GET. _____

Create and import a database from file imptid to file chid. param points to the parameter block, or can be 0. Returns +ve database ID, -ve _oserr on error. See DATAREF_doc, FSD.IMPT. _____

Include records using a user-written C function to test a record. Sbrrt is the address of a routine, or it can be 0 to act up on all records. Returns +ve selected record quantity, -1 on error See DATAREF_doc, FSD.SEL.

User written C function: char sbrrt(long dbid, void *record, struct fld_list list[], void *application) Set _oserr with any error. Note 1: the name of the routine can be anything. Note 2: the application parameter can be any pointer, and is passed to the user-defined routine. If the char is returned FALSE then that record is passed over, else it is made available. _____

Load the name list, with at least storage bytes available, on average, per name. Use 1 if the field names are not going to be altered, otherwise use enough (say 20) to allow the alteration to be stored. If storage=0, then an error occurs, use fsd_remove_names instead. Returns 0 OK, -1 error. See NAMES_doc, FSD.NMLD. _____

Find a record in which the supplied parameters match those fields which have been used to order upon. Par1 is tested against order field 1, par2 against order field 2, etc. Null parameters are ignored (=0). Only the first 8 characters of strings are used. Returns +ve record position, -1 if not found, -2 on error. See DATAREF_doc, FSD.FIND/FSD.CONT. _____

Open an existing database on the channel ID specified. It must be a file channel. Returns +ve Database ID (dbid), -ve _oserr on error See DATAREF_doc, FSD.OPEN. _____

Order a database. parameter list: field number 1 direction 1 field number 2... Note: maximum 4 entries. Note: if param_count=-1, then order memory thrown away. Note: if param_count=0, then effect as fsd_unordr. Returns 0 if OK, _oserr on error. See DATAREF_doc, FSD.ORDR. _____

Position the record pointer, and read the current record. Returns record position (+ve), -1 on error. See DATAREF_doc, FSD.POSA/FSD.POSR. _____

Set a field from the contents of fbuff. Returns 0 if OK, -1 on error. See DATAREF_doc, FSD.PUT. _____

Set a field from the contents of C string fbuff. Note: will set any type of field. Returns 0 if OK, -1 on error. See DATAREF_doc, FSD.PUT. _____

Returns the quantity of selected records. _oserr and _dberr unaffected. See DATAREF_doc, FSD.INFO. _____

Remove the specified field. Returns +ve no. of records, -1 error. See XVECTORS_doc, FSD.REMF. _____

Remove the name list memory. Returns 0 OK, -1 error. See NAMES_doc, FSD.NMLD. _____

Reset selection, and release order control block if not ordered. Return _oserr. See DATAREF_doc, FSD.RES. _____

Save the name list back to the file. Returns 0 OK, -1 error. See NAMES_doc, FSD.NMSV. _____

Use a different string compare table. -ve = Internal letter case independent table. 0 = Internal letter case dependent table. +ve = Address of user-defined table. No return. _oserr and _dberr unaffected. See SCPTR_doc; DATAREF_doc, FSD.SCPT. _____

int number)

Add a number of field markers before the given subfield. Returns +ve quantity of subfields, -1 error. See SUBFIELD_doc, FSD.SFA. _____

Get the size of a subfield. *subfield specifies either the subfield number (get), the offset (getc), or the offset of the next (getn). The offset of the subfield is returned in *subfield. Returns +ve size, -1 error.

```
int buflen, char *buffer)
```

```
int buflen, char *buffer)
```

```
int buflen, char *buffer)
```

Get the contents of the given subfield. *subfield specifies either the subfield number (get), the offset (getc), or the offset of the next (getn). The offset of the subfield is returned in *subfield. Returns +ve amount fetched, -1 error. See SUBFIELD_doc, FSD.SFGT, FSD.SFGC, FSD.SFGN. _____

Get the number of subfields in a field. Returns +ve quantity of subfields, -1 error. See SUBFIELD_doc, FSD.SFCN. _____

Get the size of the space available for a subfield. subfield specifies either the subfield number (put) or the offset (putc). Returns +ve size, -1 error.

```
int buflen, char *buffer)
```

```
int buflen, char *buffer)
```

Replace the contents of the given subfield. subfield specifies either the subfield number (put) or the offset (putc). Returns +ve amount copied, -1 error. See SUBFIELD_doc, FSD.SFPT, FSD.SFPC. _____

```
char *buffer)
```

```
char *buffer)
```

Replace the contents of the given subfield. subfield specifies either the subfield number (puts) or the offset (putc). These functions find the string length by searching for a NUL character. Returns +ve amount copied, -1 error. See SUBFIELD_doc, FSD.SFPT, FSD.SFPC. _____

Remove the subfields between from and to from a field. Returns +ve quantity of subfields, -1 error. See SUBFIELD_doc, FSD.SFR. _____

Get the number of the subfield which holds the given offset. Returns +ve subfield number, -1 error. See SUBFIELD_doc, FSD.SFR. _____

Set the order key length. This can only be done when no order or selection is imposed. An error also occurs if size is outside the range: $8 \leq \text{size} \leq 32765$. Returns 0 OK, -1 error.

Search for or continue searching for records, using a user-written C function to test a record. Returns +ve record position, -1 if not found, -2 on error See DATAREF_doc, FSD.FIND/FSD.CONT.

User written C function:

Set _oserr with any error. Note 1: the name of the routine can be anything. Note 2: the application parameter can be any pointer, and is passed to the user-defined routine. Note 3: list[1] is the first field, list[0] being irrelevant. If the char is returned FALSE then that record is ignored, else it is returned as found.

Set the name of field fld_num with the C string name. Note that this does not alter the file, and so can be applied to read-only files. Return 0 OK, -1 error. See NAMES_doc, FSD.NMST.

Remove order from a database, order memory blocks are kept. Returns 0 if OK, _oserr on error. See DATAREF_doc, FSD.ORDR.

Update the current record. Returns +ve selected record quantity, -1 on error See DATAREF_doc, FSD.UPDT.

```
short fsd_xtrl(long dbid, 0, 0, 0)
```

Get length of extra information string, and position file pointer to contents of string. Return +ve length of string, -1 on error.

```
short fsd_xtrl(long dbid, int buflen, int ftype, char *buff)
```

Get part/all of the extra information string in the file into the buffer buff, length buflen. ftype = 0: Fetch as much as possible 1: Fetch up to, and including 1st LF character (ftype=1 is used to fetch only the field names). No more than will fit into the buffer is fetched. Return +ve amount fetched, -1 on error. See ORGANIZE_doc; DATAREF_doc, FSD.XTRL.

```
long fsd_xtrs(long dbid, char *info)
```

Set the extra information string in the file to the contents of the NULL terminated C string info. Returns _oserr. See ORGANIZE_doc; DATAREF_doc, FSD.XTRS.

6.3 Global Variables

<code>extern long _database</code>

Vector base address of DATA_BIN. Set by: fsd_crt, fsd_open, and fsd_impt. —

<code>extern long _dberr</code>

Database error code. Set by most routines, but not all. —

Chapter 7

Field Name Handling

7.1 Usage

The field names can be handled through an extra set of machine code vectors. The possible operations include: fetch, set, and save.

SuperBASIC functions and procedures are supplied to handle field names too, though field names can be supplied in place of field numbers.

7.2 File Format

Note

Immediately after a database has been created, the filed name list is blank.

The field name list is stored at the beginning of the extra information string. It is formatted as:

```
"name 1", "name 2", ..., "name n"<CR><LF>user defined string
```

Note

Each individual field name is wrapped in its own set of double quotes. Single quotes cannot be used.

7.3 Memory Format

The name list is stored in memory as it is stored in the file.

The name table is stored in memory with a spare long word per field entry for the user's use. For example, this can be used for storing *SuperBASIC* name table pointers.

See DATA_IN.

7.4 SuperBASIC

CREATE/OPEN/IMPORT all load the name list, with a parameter for FSD.NMLD, register D2, that can be configured using the QJUMP Config program.

Once open, all field numbers (except in FLNAME) can be replaced by "name", however field numbers can still be used if required.

With EXPORT a string array can be used to pass field names instead of an integer array to pass field numbers.

A field name beginning with a numeric digit is treated by *SuperBASIC* as a field number.

7.5 Machine Code

7.5.1 The Main Vector

Get the name handling vector base address.

FSD.NAME	JSR	\$84 (An)
Call Parameters	Return Parameters	
D1	D1.L	Preserved
D2	D2.L	Preserved
D3	D3.L	Preserved
A0	A0.L	Preserved
A1	A1.L	Preserved
A2	A2.L	Vector base address
Error return:		
ERR.NI	Not implemented	

This routine has to be called to get the name handling vector base address, which is not the same as the normal vector base address.

The extra field name handling vectors are:

- #### 7.5.2.1 FSD . NMLD - Load Field Name List

If D2=0 then the name table and name list are released, and the routine returns. The contents are not kept.

D2 gives the minimum average storage length per field. The vector makes allowance for "", or ""<CR> per field and the <LF> on the end. Since D2 is an average, names can be longer or shorter than that.

$$Field\ count * (D2 + 3) + 1$$
$$Field\ list\ size + (Field\ count * 2)$$

When this vector is called, the first part of the extra information (see `FSD.XTRS`, `FSD.XTRL`) is loaded in as the field name list, up to the first LF. It is then parsed to fill in the name table. If it does not parse correctly, the field name list is replaced with “”, “”, “”, “”<CR><LF>.

If a field name list exists when the `FSD.XTRS` vector is called, then this vector is re-issued at the end of that vector, if no errors occur.

This vector will also work for a directory database, using the set of names stored in DATA_BIN.

7.5.2.2 FSD.NMSV - Save Field Name List

FSD.NMSV JSR \$08(An)			
Call Parameters		Return Parameters	
D1		D1.W	Length saved
D2		D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1		A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.NF	No name list loaded		
ERR.NI	Operation not available (Eg: a directory database)		
ERR.OV	The name list will not fit in the database		

This vector saves the name list into the database file if possible. It overwrites the old name list in the first part of the extra information. However, it does not affect the remainder of the extra information string.

The name list must fit into 32767 bytes minus control information, code section, and user defined string. If it does not fit, ERR.OV is returned.

7.5.2.3 FSD.NMGT - Get a Field Name

FSD.NMGT JSR \$0C(An)			
Call Parameters		Return Parameters	
D1.B	Field number	D1.W	Amount fetched / available
D2.W	Buffer size / 0	D2.L	Preserved
D3		D3.L	Preserved
A0.L	Database ID	A0.L	Database ID
A1.L	Buffer pointer	A1.L	Preserved
A2		A2.L	Preserved
Error return:			
ERR.NF	No name list loaded		
ERR.OR	Field number out of range		

If D2 is 0, then the amount available is returned in D1. Otherwise this vector gets the name of the given field, and puts it into the buffer, to a maximum of D2 characters.

7.5.2.4 FSD.NMST - Set a Field Name

```

FSD.NMST JSR $10(An)
Call Parameters      Return Parameters
D1.B   Field number  D1.W   Amount copied
D2.W   Buffer size    D2.L   Preserved
D3     D3.L   Preserved
A0.L   Database ID   A0.L   Database ID
A1.L   Buffer pointer A1.L   Preserved
A2     A2.L   Preserved

Error return:
ERR.NF   No name list loaded
ERR.BO   The field name will overflow the name list
ERR.OR   Field number out of range

```

This vector puts the contents of the buffer into the name list in place of the old name. However, if the new name will not fit, no data is copied, and the old name remains.

Note

This vector does not update the on disc database file, that must be done by calling FSD.NMSV.

7.5.2.5 FSD.NMFN - Search Name Fields

```

FSD.NMFN JSR $14(An)
Call Parameters      Return Parameters
D1     D1.B   Field number/0
D2.W   Buffer length D2.L   Preserved
D3     D3.L   Preserved
A0.L   Database ID   A0.L   Database ID
A1.L   Buffer pointer A1.L   Preserved
A2     A2.L   Preserved

Error return:
ERR.NF   No name list loaded

```

If it can, this vector matches, case independently, the name in the buffer to one of the names in the name list, and returns the field number.

If no match is made, 0 is returned.

Chapter 8

String Comparison Tables

8.1 Introduction

The String compare "translate" table, is used to change how strings are compared. Example uses:

- Implementing letter case (in)dependence.
- Rearranging the order to allow for foreign character sets.

Two 256 byte tables are supplied inside DATA_BIN. The summarised sort order is:

```
SPACE
!"#$%&'()*+,-/:;
<=>?@[\\]^_
POUND Symbol
{|}~
COPYRIGHT Symbol (c)
DOT
0123456789
AaBbCcDdEeFfGgHhIiJjKkLlMm
NnOoPpQqRrSsTtUuVvWwXxYyZz
characters 0--31
characters 128--255
```

See QDOS machine code vector UT.CSTR (types 0,1) in the QDOS/SMSQ Reference Manual.

8.2 SuperBASIC Usage

To set the pointer:

```
SCPTR #3(;pointer)
```

If no pointer is supplied, SCPTR #3, then 0 is used. If pointer is zero, a case dependent compare will be used. This is the default.

If pointer is negative, a case independent compare will be used.

These BASIC procedures are affected:

- EXCLUDE
- INCLUDE
- LOCATE
- LOCATEC
- ORDER
- SEARCH
- SEARCHC

Note: FIND and FINDC are not affected by this table.

8.3 Machine Code Usage

To set the pointer:

```
move.l databaseID(A6),A0
lea pointer,A1
JSR FSD.SCPT(A2)
```

The above example assumes that:

- The database ID is stored in dataspace, pointed to by the A6 register.
- “Pointer” is a label at the start of a SCPTR table in memory.
- The main appDBAS system vector is stored in register A2.

These machine code vectors are affected:

- FSD.ORDER
- FSD.SEL
- FSD.FIND with FSDF.FOR
- FSD.CONT with FSDF.FOR
- FSD.FIND with FSDF.FSE
- FSD.CONT with FSDF.FSE
- FSD.COMP

Notes: FSD.FIND with FSDF.FIN and FSD.CONT with FSDF.FIN are not affected by it, as they use a different routine.

A vector is supplied to use the tables to compare user supplied strings (see FSD.COMP).

8.4 The Tables

Each SCPTR table is a block of 256 bytes. Each byte corresponds to one character. All 256 bytes can be the same value, or any or all can be different.

Every byte fetched from a string that is to be compared is used as an index to the table.

Internally, the table is used in a manner similar to the following:

A byte would be fetched from a string and indexed into the table to get another value:

MOVE.L	CMP.TABL(PC),A2	Get the address of the table
MOVE.B	(A1)+,D1	Get a byte from the string at (A1)
ANDI.W	#\$00FF,D1	Make it into a word
MOVE.B	0(A2,D1.W),D1	Index into the table

A byte would be fetched from another string and would then be indexed, eg:

MOVE.B	(A3)+,D2	Get a character
ANDI.W	#\$00FF,D2	Make it into a word
MOVE.B	0(A2,D2.W),D2	Index into the table again

And finally compared, eg:

CMP.B	D1,D2	Compare the characters
BEQ.S	eeee	Jump if equal-to
BGT.S	gggg	Jump if greater-than
BLT.S	llll	Jump if less-than

At each position in the table is a byte code which determines the position in the sort order for the ASCII character with that character code.

For example, if you wanted the letter "A" to be the lowest in the series, then at \$41 (the ASCII code for "A") into the table, you would put a \$00; and if you decided to have case independence then at \$61 (the ASCII code for "a") you would also put \$00.

You might not want the letter B next in the order though, but if you wanted "*" next to "A" for some reason, put \$01 into the table at \$2A (the ASCII code for "*").

8.5 Modifying DATA_BIN and DROM_BIN Tables

There are two tables in DATA_BIN and DROM_BIN:

The first is located in the 256 bytes immediately following the string "STRING COMPARE TABLE 0"; this is the table used by default, and when a zero pointer is supplied to SCPTR or FSD.SCPT.

The second is located in the 256 bytes immediately following the string "STRING COMPARE TABLE M"; this is used when a negative pointer is supplied to SCPTR or FSD.SCPT.

These tables can be modified at will, by directly overwriting them in the file.

Warning

Changing the in-file tables will affect *all* programs attempting to use these tables. Generally, it is better to use the facility for supplying a user-defined table.

Chapter 9

Directory Support Using Database Vectors

9.1 Associated Files

DDIR_BAS is an example program. It prints an alphabetically ascending directory of the disk in FLP2_ to #1.

9.2 Usage

There are two ways of accessing this facility. The first is through a basic procedure (OPEN_DDIR) and the second is through a machine code vector (FSD_DDIR). All these routines do is to set up an appropriate set of database memory blocks. The database is then accessed and manipulated in the usual way.

9.2.1 Fields

To aid access, eight fields are defined:

Type	Length	Name	Contents
2	-	LENGTH	File Length
1	-	TYPE	File access *256+ file type
2	-	DATA	Dataspace size - EXECutable type
2	-	OFFSET	GST Linker "OFFSET" value
0	36 char	NAME\$	File name
2	-	UPDATE	Update date
1	-	VERSION%	Version number
1	-	FILE%	File number
2	-	BACKUP	Backup date (not used)

9.2.2 Records

Records are available as usual, however there may be numerous blank entries which can be suppressed by:

```
EXCLUDE #C; 5, "=", ""
```

9.2.2.1 Modifications and Extra Information

Certain procedures and functions are suppressed:

- APPEND
- REMOVE
- SEXTRA

And the equivalent vectors:

- FSD.APPN
- FSD.DEL
- FSD.XTRS
- FSD.CDLLD (Returns D1=0 when D2=0, else it returns ERR.NI)
- FSD.CDSV

However UPDATE and FSD.UPDT are available and should be used with *extreme* care.

9.2.3 Other Access

However, all other applicable procedures, functions and vectors are available for use. They are:

- CLOSE_DATA, FSD.CLOS
- COUNT(),
- EXCLUDE,
- EXPORT, FSD.EXPT
- EXPORT_O, FSD.EXPT
- FETCH(),
- FEXTRA(),
- FIND, FSD.FIND
- FINDC, FSD.FIND
- FLLEN(),
- FLNAME(),
- FLNFIND(),
- FLNUM(),

- `FLTYP()`,
- `FOUND()`,
- `INCLUDE`,
- `LOCATE`,
- `LOCATEC`,
- `ORDER, FSD.ORDER`
- `RECNUM()`,
- `RPOSAB, FSD.POSA`
- `RPOSRE, FSD.POSR`
- `SCPTR, FSD.SCPT`
- `SEARCH, FSD.COMP`
- `SEARCHC, FSD.COMP`
- `SET`,
- `SLNAME`,
- `UPDATE, FSD.UPDT`
- `FSD.CONT`
- `FSD.GET`
- `FSD.INFO`
- `FSD.NAME`
- `FSD.OINF`
- `FSD.PUT`
- `FSD.RES`
- `FSD.SEL`
- `FSD.XTRL`

All name handling machine code vectors apart from `FSD.NMSV` still work as normal.

Also `FSD.UPF` and `FSD.DNF` are available, but should NOT be used on a directory, as this would destroy the directory, whose elements are position dependent.

Chapter 10

Manipulate A Database

ALTER_BIN is a utility program for use with the *DBAS* package. It requires DATA_BIN or DROM_BIN installed beforehand. It cannot manipulate Archive type database files.

ALTER_BIN is an EXECutable type job, it is intended to fulfill the role of ALTER/INSERT in *Archive*. The Extra Information String must contain the field names in the format specified in Chapter CHAPTER: ORGANIZE - **DBAS System Organization**, which begins on Page 7.

The job allows viewing, adding, amending, deleting and finding records. Instructions are supplied as the program is used.

For those with Toolkit II the program can be made to operate on an already opened database from BASIC.

This program recognises the contents of a database code section, and calls the appropriate routine parts. See ?? - ?? on Page ??.

Directory databases can be supplied to ALTER_BIN, but can not modified.

10.1 Use From SuperBASIC

To get the job to ask for a filename to open:

```
EXEC flp2\_ALTER_BIN
REMark Or:
EXEC_W flp2\_ALTER_BIN
```

To use an already open database if channel is known:

```
EX flp2\_ALTER_BIN, #C: REMark Where C is the channel number.
REMark Or:
EW flp2\_ALTER_BIN, #C
```

To use an already open database if address is known. The address can be found with the DBADDR function supplied:

```
EX flp2\_ALTER\_BIN, "@#dddd"
REMark Or:
EW flp2\_ALTER\_BIN, "@#dddd"
```

Where “dddd” is the decimal address.

Or:

```
EX flp2\_ALTER\_BIN, "@*nnnn"
REMark Or:
EW flp2\_ALTER\_BIN, "@*nnnn"
```

Where “nnnn” is a four-byte string in which an address is stored as if by POKE_L. This facility is intended for use from jobs other than *SuperBASIC*.)

When an already opened database is used, any ordering and selection stay in effect. In addition, the record displayed initially is the current record upon handing over control. Likewise the record displayed on quitting is the new current record.

10.2 Use From Machine Code or C

The command string should be left as null if no database is to be supplied.

To use with an already opened database from machine code or C, the command string must be set with one of the two following options:

1. “\$@*nnnnnn”—Where “nnnnnn” is the address. The string length for the address must be six bytes. For example, if the database address is \$40720 then \$402A00040720 would be stacked as the command string. \$402A = “@*”.
2. “@#dddd”—Where dddd is the decimal address. Using the above example, then “@#263968” would be stacked as the command string. \$40720 = 263968.

Any channels supplied are ignored.

10.3 Source files for ALTER_BIN

- ALTR_ALIN_ASM
- ALTR_CSEC_ASM
- ALTR_DATA_ASM
- ALTR_FIND_ASM
- ALTR_IN
- ALTR_LINK
- ALTR_MAIN_ASM
- ALTR_SBRT_ASM
- ALTR_SCRN_ASM
- ALTR_WDEF_ASM

Extra source files required for main package:

- DATA_IN
- FAST_MATH_ASM
- QDOS_DATA_IN
- QDOS_TRAP_IN
- QDOS_VECT_IN

Appendices

Appendix A

Installation of Printer Drivers

A.1 Associated Files

INSTALL_DBS The installation data and program. It requires **ALTER_BIN** to use it. Just execute **ALTER_BIN** and pass it this filename. **ALTER_BIN** knows how to handle the rest.

INST_BIN The checking and installation code. This is put into the code section (See Chapter ?? - ?? on Page ??) of **INSTALL_DBS** with **CODE_SAVE_BIN**.

INST_IN The source files for **INST_BIN**.

INST_MAIN_ASM

INST_PCHK_ASM

INST_INST_ASM

INST_BAS Run this basic program to convert your **INSTALL_DAT** to records in **FLP2_INSTALL_DBS**.

A.2 Usage

From **ALTER_BIN**, edit the records to suit your printer (some example records are included), note that **INST_BIN** checks most of the strings you type in, and will complain if an invalid item is input. See below. NOTE also: the **FILENAMES**'s may well not be appropriate to your *QUILL*, *ARCHIVE* and *ABACUS*, as I have tailored mine to work from one disk with separate driver files.

Then from the main menu press F3. This will take you to a secondary menu. To configure the installation drive name, press "C", and then follow the instructions. To see the further options, press "O", and another menu should appear in a different window: To see a list of what valid items can be typed into which fields, press "H". To install the current record, press "I".

A.3 Valid Items

Most of the fields are validated after editing to make sure that the constituent items are valid.

- FILENAME\$: No checking.
- DRIVER\$: No checking; but cut down to 9 characters.
- COMMENT\$: No checking.
- PORT\$: No checking; but cut down to 17 characters.
- BAUD%: Must be: 9600, 4800, 2400, 1200, 600, 300, 75.
- PARITY\$: Must be: NONE, SPACE, MARK, ODD, EVEN.
- CONTINUOUS\$: Must be: YES, NO.
- PAGE_LENGTH%: Must be: 0-255. (Lines)
- WIDTH%: Must be: 0-255. (Characters)

All other fields must have 0–11 items. Where each item is one of:

1. Decimal code: 0–255
2. Hex code: \$0–\$FF
3. Character: "x or 'x
4. An ASCII control code:
 - NUL SOH STX ETX EOT ENQ ACK BEL BS HT LF VT
 - FF CR SO SI DLE DC1 DC2 DC3 DC4 NAK SYN ETB
 - CAN EM SUB ESC FS GS RS US DEL

These items must be separated by commas.

A.4 Source Files

The source files are provided as an example of the use of a code section. Chapter ?? - ?? on Page ?? describes the use of a code section, both from the point of view of the code section and the calling routine.

Appendix B

Archive Replacement

This chapter is the original article that David Howells wrote for the Quanta Magazine, explaining about the new *DBAS* system.

B.1 History

After doing a large piece of programming in Archive, which I found unwieldy, I decided to write my own database handler. This is written in machine code, but is usable from machine code, Basic, and C. I have submitted this to the QUANTA library, and it should be available in due course.

B.2 Overview

The following sections detail the overview of the *DBAS* database system.

B.2.1 General

- It can be used with any directory device driver, eg: MDV, FLP, N, WIN, and RAM.
- It can IMPORT/EXPORT Archive/Abacus type export files.
- The vast majority of Archive's ability has been implemented both in Basic and machine code, and plus some extra facilities.
- Supported field types are:
 - Word integer: 2 bytes
 - Long word integer: 4 bytes

- Floating point: 6 bytes
- Fixed storage length string: maximum 32765 characters
- Variable storage length string: maximum 32765 characters
- Variable length strings are more efficient on space, fixed length strings are more efficient on time.
- Database system maxima:
 - Record length 32767 bytes
 - Record quantity 32767
 - Field quantity 255
- Records can be added, amended, removed, and retrieved.
- The way strings are compared can be altered. Eg: usually $1 < 2 < 3 < 4$ is true, but this can be altered, eg to: $1 < 3 < 2 < 4$. This can be configured differently for every database open. The package itself contains two default tables for this purpose, which can also be configured.
- Routines can be added to a database to perform special functions.

B.2.2 Superbasic

- The package can open databases and attach them to *SuperBASIC* channels.
- It uses the `DATA_USE` string if available, and if required.
- It attaches its memory blocks to *SuperBASIC* so that they can be removed with `CLCHP` (Toolkit II). A separate `CLCHP` program is supplied for those without Toolkit II.
- A procedure is supplied to close a channel and remove its associated memory.

B.2.3 Machine code

- A new `TRAP #3` option is "attached" to each directory device driver to allow a base address to be fetched.
- All other routines are accessed relative to this base address.
- Absolute and A6-relative addressing are supported (using `TRAP #4`)

B.2.4 C68 facilities

- Library, include, and documentation files are supplied.

B.3 Implementation

The database handler is in two parts, a set of machine code vectors, and a Basic interface. A ROMable file containing both parts is also supplied.

Various demonstration programs in both Basic and machine code, and a basic utility are included.

David Howells
28, Medlock Crescent,
Handsworth,
Sheffield.
S13 9BD

Appendix C

Updates

C.1 Changes to Database Handling

(Changes to DBPTR_BIN follow at the bottom)

C.1.1 Version 2.00

- Names handling introduced into the vectors, Basic interface, and C interface.
- The `FSD.ORDER` (`ORDER`) algorithm was rewritten to be a lot faster.
- Array parameters made available to Basic procedures.
- When the comparison function is called from the C68 `fsd_search`, `fsd_incl`, & `fsd_excl` functions; the first field descriptor is at `fld_list[1]` not `fld_list[0]`.

C.1.2 Version 2.01

- The memory allocating routines now allocate memory with the owner as the job which opened/created the database. The user memory interface routines have the owner Job ID handed over in D2.
- Some minor bugs corrected.

C.1.3 Version 2.02

- `DATA_BIN` & `DBAS_BIN` now display messages when called. `DATA_BIN` prints to QDOS channel \$00000000, and `DBAS_BIN` prints to channel #0 of the Basic in which it is called.

- `DATA_BIN` has a `QJUMP` config style configuration block for default order key length and compare case dependence, as well as for activating the database "thing".
- All procedures in the *SuperBASIC* interface now have a function version (preceded by `FDB_`) that returns the error code as a float.
- Reordering of records after appending & updating now proceeds with the same algorithm as ordering, and so is alot faster.
- `FSD.ORDER (ORDER)` now preserves the select status of the records. Note, however, that unordering the database no longer releases the memory, but must be followed by a call to `FSD.RES (RESET)` to do so.
- The size of the order string keys can now be increased through use of `FSD.SOKY (SORDKEY)`.
- `FSD.SEL (INCLUDE & EXCLUDE)` now check the records in file order rather than any imposed order, and so go faster.
- `FSD.SEL & FSD.FIND-FSD.FSE (INCLUDE, EXCLUDE, & SEARCH)` now can take a Contains string condition, and have a flag to NOT the result. Also "AND" linkages take priority over "OR" linkages.
- The record expansion routine (for dynamic records) has been rewritten, and so is now faster. Therefore `FSD.FIND & FSD.SEL (FIND, LOCATE, SEARCH, INCLUDE & EXCLUDE)` go faster, as well as record positioning & exporting.
- `OPEN_DDIR` now loads the inbuilt field name table.
- A new database editor program has been added that requires the pointer environment to run. However, its source is very long, so its sources and the `ALTER_BIN` sources are to be submitted on another disk.

C.1.4 Version 2.03

- The v2.01 memory allocation improvement is now documented.
- The machine code test program `TEST_MC2` now allocates memory with the supplied job ID.

C.1.5 Version 2.10

- The `TRAP #4` flag is now tested & cleared on SMS2 systems.
- A new set of vectors to manipulate subfields within string fields has been added & documented (Basic commands begin `SUBF_`).
- A new pair of vectors to add and remove fields has been added (`ADD_FIELD` and `REMOVE_FIELD` in *SuperBASIC*).
- The `DBADDR` function has now been transferred to `DBAS_BIN`, instead of having its own files.
- The Basic function `FETCH` should now work properly.

- A new vector has been added to copy the selected records, in order, from an open database to a new database file. (COPY_DATA in Basic)

C.1.6 Version 2.11

- FSD.GET (FETCH) now returns ERR.BO rather than ERR.OR if the field contents will not fit in the buffer.

C.1.7 Version 2.12

- FSD.FIND & FSD.CONT (FIND, FINDC, SEARCH, SEARCHC) now find only the matches they're supposed to.
- FSD.ORDER (ORDER) and FSD.FIND & FSD.CONT (LOCATE, LOCATEC) now do comparisons properly (the alphabet no longer seems to start at "P"!).
- A new version of the C68 library (LIB_libdbasdu_a) has been added. This uses extra underscores, accessing __oserr and __dberr which the latest versions of C68 require.

C.2 Changes to DBPTR_BIN

C.2.1 Version 1.01

- Two new display modes have been added: one uses as much screen as is necessary to display all of a field's content (if possible); the other mode displays field information.
- Newline characters within a field are displayed as down-and-left arrow characters, like <ENTER> on the keyboards of some computers. These can be placed in a string whilst editing with shift-enter.
- The help facility has been improved. Instead of pages, a continuous stream of text is available on each record. It is also now possible to click on the text in square brackets, to transfer that to the key box.
- Also, the editor source disk has been updated.

C.2.2 Version 1.02

- The problem with digits being repeated in integer fields has been corrected (this depends on the QDOS version being used).

- The info option now displays the program version number.

C.2.3 Version 1.03

- The help application subwindow (help text display) now works properly after the help window has been moved.