CERN - DATA HANDLING DIVISION
DD-73-17
M.C. Turnill
June, 1973

TABMAN : A Dynamic Core and File Management System

(Revised version of paper presented at ECODU-15 (Control Data European Users Conference), Bled, Jugoslavia, April 1973).

1. Introduction

TABMAN is an integral part of a generalised Information Retrieval System (TABLOID) that is being developed to meet the needs of CERN II in constructing the new 300 GeV/c accelerator. Before discussing the details of TABMAN and its functions it is essential to provide some background concerning the requirements for TABLOID and an overview of the TABLOID design, which will be more fully described in a subsequent report.

During the construction period over 8 km of tunnels will be dug including the main ring. The equipment needed to operate the accelerator will require some 30,000 cables to be installed. These cables have a wide variety of functions - power, control signals, computer data links and television.

TABLOID has been designed, following CERN II specifications[1]. to provide all the working documents necessary to install, modify and use the cables in the accelerator. Reports will be generated for two different classes of people, firstly for the managers controlling the project and secondly for the technicians carrying out the installation work. In the first case the reports would contain costing and control information, whilst in the second case the technicians will be given work sheets indicating the types and lengths of cables to be drawn from the stores and the two locations between which the cables should be laid. Examples of these two types of reports are shown in Figs. 1 and 2 and an example of a user report request in Fig. 3.

Each cable has associated with it at least 20 different items of information such as type, length, position, job number, and so on. It is possible for some items to have several occurrences. For example two "positions" may appear in one record. Such multiple items are known as "repeating groups" which implies that the total number of items in a record may vary. The requirements of the above project are that reports may be requested based on almost any combination of a large number of key items - a typical multi-key situation. In addition the report writer contained in the system must be capable of accessing numerous "dictionaries" that are to be supplied as separate files.

After an initial appraisal of existing CDC data management software products we decided to write our own system since none of them currently meet the necessary requirements. In particular the features for handling dictionaries were inadequate and none contained sufficiently general report writers.

Although TABLOID was designed with the cables problem in mind, it is completely general within certain limitations and will in fact be used for several other applications such as maintenance scheduling and indexing drawing files.

SPS CONTROL INSTALLATION EXPENDITURE / 1. COSTS PER JOB

JOB NR.	80001	BMC1
CODE NR.	6134/345	M. BENOIT
SYSTEM NR.	2301	TELEVISION
WORK PERIOD	13.02.72 -	15.07.72

MATERIAL COSTS

CABLES

...ABBN...	.....TYPE.....	METER	PRICE PER METER SFR	AMOUNT SFR	TOTAL SFR	FINAL TOTAL SFR
MB10	MULTICORE 10 x 0,5 SQMM	2000	6.50	13.000.00		
MB18	MULTICORE 18 x 0,5 SQMM	3000	10.00	30.000.00		
MB48	MULTICORE 48 x 0,5 SQMM	5000	15.00	75.000.00		
CA50	COAXIAL RG 316 U 50 OHM	10000	2.00	20.000.00	138.000.00	

CONNECTORS

...ABBN...	.....TYPE.....	PIECES	PRICE PER PIECE SFR	AMOUNT SFR	TOTAL SFR	
B12SF	BURNDY 12 PIN SOCKET FEMALE	50	30.00	1.500.00		
B12PM	BURNDY 12 PIN PLUG MALE	100	35.00	3.500.00		
B50SF	BURNDY 50 PIN SOCKET FEMALE	100	80.00	8.000.00		
CB50PM	COAXIAL BNC 50 OHM	1000	2.00	2.000.00	15.000.00	

OTHER MATERIAL

...ABBN ...	.....TYPE.....	UNITS	PRICE PER UNIT SFR	AMOUNT SFR	TOTAL SFR	
PA 6U28B12	PANNEAU 6 UNITES 28 BURNDY 12 BROCHES	20	70.00	1.400.00		
PA 1U 7B12	PANNEAU 1 UNITE 7 BURNDY 12 BROCHES	60	10.00	600.00		
PA 6U50C	PANNEAU 6 UNITES 50 CONNECTOR COAXIAL	100	40.00	4.000.00		
G.RET 3,2	GAINE RETRACISSANTE NOIRE	100	1.00	100.00		
AT KBW 302	ATTACHES TRANSPARANT KBW 302	500	3.00	1.500.00	7.600.00	160.600.00

MISCELLANEOUS (SELF SERVICE, DRAWING OFFICE, ETC.)

10% OF FINAL TOTAL				16.060.00		176.660.00

Figure 1

SPS CONTROL CABLE LIST 5 / INSTALLATION LIST

SYSTEM NR. 3011 TELEVISION
JOB NR. 80001 BT 10

CABLE TYPE MB10 MULTICORE 10 x 0,5 SQMM

CABLE ..NR..	CABLE CONNEC	.....FROM..... BUILD. EQPT + POS SUFF CHASSIS	.....TO..... BUILD EQPT + POS SUFF CHASSIS	CABLE CONNEC	LGTH (M)	.....REMARKS.....
345671	B12PM	BA.41. RA..3456.5 A..6 3901	BA.50. RA..736410 C.15 3901	B12PM	1345	XXXXXXXXXXXXXXXXXXXXXXXXXX
345672	B12PM	BA.41. RA..2345	BA.50. RA..3452..	B12PM	1345	
345673	B12PM	BA.41. RA..345212 C.12 3910 10	BA.50. RA..231420 B..5 3910	B12PM	1345	
345674						
345675						
345676						
335677						

TOTAL LENGTH OF CABLE 10534

CONNECTION ABBREVIATIONS

		TOTAL
B12PM	BURNDY 12 PIN PLUG MALE	12
B12SF	BURNDY 12 PIN SOCKET FEMALE	20

LOCAL ABBREVIATIONS

ABBN BUILDING		ABBN EQUIPMENT		ABBN SUFFIX	
BA	AUXILIARY BUILDING	RA	RACK	RX	BOX
BT	BUILDING RING TUNNEL	MAD	MAGNET DEFOCALISING		

CHASSIS NR.	TITLE
3901	TIMING PULSE REPEATER
3910	B PULSE GENERATOR

Figure 2

```
*DATABASE     PFN=CABLESBASE,ID=OWNER,PW=RSSX,CY=3

*INDICT

ABA     BUILDING, AUXILIARY(1-6)

ATT 10 TRANSFER TUNNEL, INJECTION

D1205  VACUUM / TURBO-MOLECULAR PUMPS          50001 TO 5200 73000 TO 74500

E80001 PROTOTYPE WIRING              5101SAG8052/304150372
     .
     .
     .
*DATAREAD

1  8701CB50   2051B50PMBA  10 RA  171520A  12575  TS 1 MBA 1103 RX 1   1B50SF
2  870114028O019K0F4172CONTROL SIGNAL            A137BA159CB148A
-------------------------------------------------------------------------------
1  8702CB50   2251B50PMBA  10 RA  171520A  22575  TS 1 MBB 1109 RX 1   1B50SF
2  870214028O019K0E4172CONTROL SIGNAL            A137BA159CB149A
-------------------------------------------------------------------------------
1  8703CB50   2381B50PMBA  10 RA  171520A  32575  TS 1 MBB 1113 RX 1   1B50SF
2  870314028O019K0E4172CONTROL SIGNAL            A137BA159CB150A
-------------------------------------------------------------------------------
     .
     .
     .
*LISTS
2          8      100001 TO 150000
3          1      3101
4          9      BT 121MGF
*STOP
```

Figure 3: Users Report Requests

Notes:
i) The *DATABASE card introduces a random access file containing the schema definitions. This file will itself refer to the appropriate SIS file.

ii) Following the *INDICT card are new dictionary definitions. The first column identifies the dictionary.

iii) Following the *DATAREAD card are new cable cards which are to be inserted into the data base.

iv) Following the *LISTS card are users report requests.

2. An Overview of TABLOID Design

Previous experience on this type of problem for the ISR cabling convinced us at an early stage that we should design a system that was reasonably data independent, since the users requirements change very frequently. This led us to separate the data definitions from the program and for this we have used the term "schema". This "schema" approach is also used by the CODASYL committee in its recent recommendations on Data Base Design[2]. In our case this implies an interpreter in between the users report requests and the accesses to the data base. Each record type in the data base has a schema description which names the items in the record together with their characteristics. A typical schema description might appear as in Fig. 4.

These schema tables are read by TABLOID during an initialisation run, validated and stored as a secondary data base. The way in which such schemata can be associated with the data bases will be mentioned later.

The data bases maintained by TABLOID are normally SCOPE Indexed Sequential (SIS) files. Each record in such a file has a "key" which consists of one or more items in the record concatenated together. Any record in the data base may be retrieved by either part or all of this key. This type of file organisation is described in more detail in Section 4.

These schemata are accessed using an interpretive language which currently has rather rigid syntactic rules which may be relaxed at a later stage. An example of the type of coding that can be performed in this language is shown in Fig. 5. Each function has the following form:

<Label>|<Function>|<4 input parameters (up to)>|<2 output parameters>

The input parameters may refer to items named in the appropriate schema descriptions.

The first output parameter is a logical result .TRUE. or .FALSE. and the second output parameter is a numerical result (if any). The language is punched for the time being on fixed field column card boundaries.

The language is "compiled" into internal tables. That is to say the functions and literals which appear in the language are translated into numerical values and pointers. This is a process that need take place only once (assuming that the original coding is correct!) and the resulting internal tables may be stored on a file ready for use at execution time. The tables generated during this phase all contain relative pointers, i.e. literals are referred to

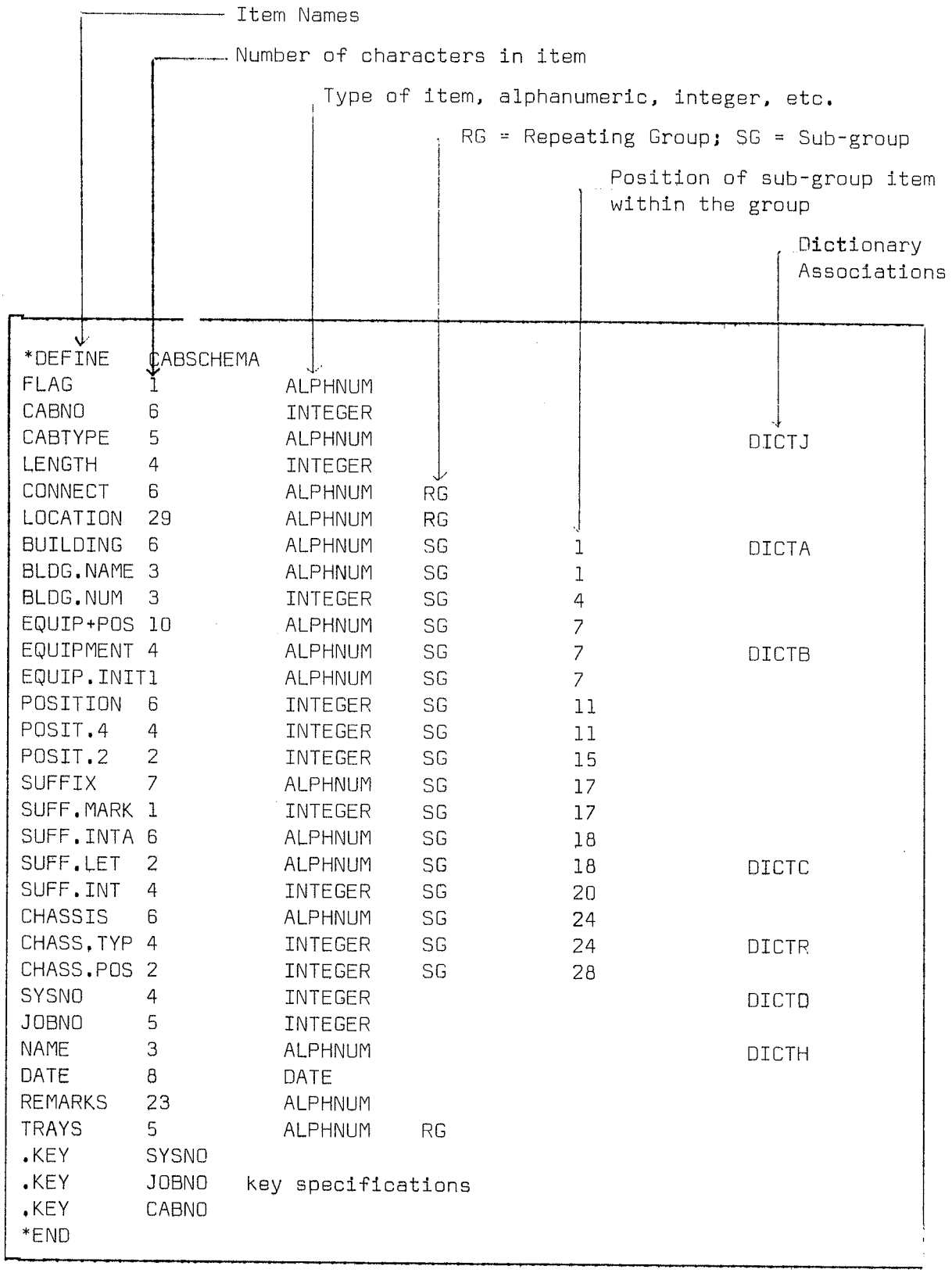


Figure 4: Schema Description for Cable Cards

Note: In the key specification the order specifies the concatenation of items that are to be used to construct the SIS key.

```
SECTION RETRIEVE
        LOOPDB      L1
        NORANGE     SYSNO    LIT1   LIT2   L1
        SORTKEY     SYSNO
        SORTKEY     CABNO
L1      ENDLOOPDB
*END
```

Figure 5

Example of schema language used in Retrieval section

Notes: i) The item names SYSNO, CABNO can be found in the schema definition - see Figure 4.

ii) LOOPDB is like a "DO loop" in FORTRAN where ENDLOOPDB is the end of the loop. These instructions say: loop over the data base checking whether the system number (SYSNO) lies in the range LIT1 to LIT2. If not in range go to L1, otherwise accept the record and indicate that the sort keys are (SYSNO+CABNO). On passing through the end of the loop (i.e. end of file encountered) the retrieved file is automatically sorted on system and cable number. In this example the entire SIS file is searched.

as a relative address from the start of a given literal table.

There is also an "executor" which actually executes the instructions which have been laid down in these tables. Since the internal tables have been "relocated" to contain absolute pointers before "execution" (or "interpretation") begins this process is reasonably fast.

As mentioned earlier it is possible to associate dictionaries with various items in the records in the main data-base. This association is done through the schema description. These dictionaries themselves have schemata, although rather straightforward ones, and are maintained by TABLOID exactly as any other file would be.

TABLOID is programmed largely in FORTRAN using a library of Compass coded routines for special purpose functions. The choice of language was largely inherent in the CERN environment but the decision was taken in the belief that our design would be an efficient one and that no loss of efficiency would be occasioned by our non-use of COBOL on a CDC 6400 which has no character instructions (COMPARE/ MOVE unit).

Using this interpretive language we are able to build up the four classical modules needed for a system of this type, that is to say:

i) Initial Input and Validation

ii) Retrieval and Sorting

iii) Updating

iv) Output or Report Writing.

3. Constraints Imposed by the Operational Environment at CERN

Ideally TABLOID should be usable either in batch or interactive mode. The first version concentrates on batch mode, but later versions will run interactively under INTERCOM.

CERN has as its central computing facility a CDC 7600 coupled to a CDC 6400 using a fast link of 400 Kc/s. All major peripherals are attached to the 6400 including the principal permanent file storage medium, the CDC 841/844 disk packs. The only significant peripherals attached to the 7600 are two large fixed head disks (7638's). These disks are extremely unsuitable for random access, and permanent file usage on them is discouraged. Additionally CDC provide SIS only as a CDC 6000 product and thus we are constrained to use the front-end. However this front end has only 200 K_8 of central memory together with 2000K_8 words of Extended Core Storage (ECS). This

additional storage may not contain executable code and is thus considered as a very fast peripheral device. For efficiency, data should be transferred to and from ECS in blocks rather than as single words. The SCOPE operating system itself makes heavy use of ECS.

The CERN front-end has to cater for all I/O to and from the 7600 as well as the remote batch stations and interactive users on a variety of terminals. The "fast station" which links the two machines currently occupies up to 40 K_8. When the other constraints of system occupancy and Intercom usage are taken into account this leaves in practice a maximum user job size of 100 K_8 using SCOPE 3.3. Under SCOPE 3.4 this figure is expected to be considerably lower.

4. TABMAN (Table Manager)

4.1. Introduction

An information retrieval system such as TABLOID makes widely fluctuating demands on the core occupancy depending upon the user's report requests. Unlike normal FORTRAN programs the number of active files can also vary widely during a job and it is essential to be able to open and close files dynamically during execution to save unnecessary buffer space being allocated. Thus at the outset there was a strong argument for building a central memory management system. Thus, a self contained package, TABMAN, was developed which maximises the use of the available hardware to offset the shortage of central memory space. At the same time TABMAN allows ease of access to the three primary file types - sequential, random access (word-addressable) and Scope Indexed Sequential (SIS) files.

Extended Core Storage (ECS) and random access disk files provide a second level back-up when central memory becomes full, without the user being aware of their existence. In addition TABMAN also controls the placing of overlays in ECS to increase execution efficiency.

4.2. Concept of a Table

The basic unit of storage allocation in central memory (CM) is known as a "table". This is essentially a two dimensional array with m entries (or records) of length n. The entry length of a given table is always fixed. All CM storage is requested in terms of tables and all allocations are made in Blank COMMON (called Q). There is no other structure to a table except that the first (or zero'th entry) is considered special, and contains a mask that may be used for search purposes (see 4.5.). Thus a table with an entry size of 4 words and containing 6 entries occupies 28 words and appears as in Fig. 5. No table can exceed the total size of blank COMMON. TABMAN recognises

only tables without regard to their contents. For each table in Q an entry is made in a "Master Table" which resides at the high address end of Q. By convention Q is always fully allocated with tables; that is to say all space is accounted for either as active tables, or as "dead" unused tables. At initialisation time Q consists of one large dead table plus the Master Table.

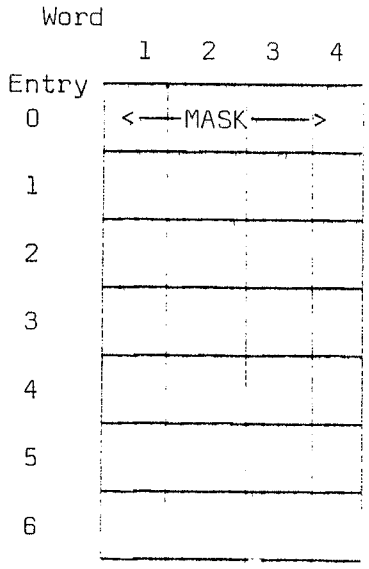


Figure 5

During execution Q may well have an appearence like that shown in Fig. 6.

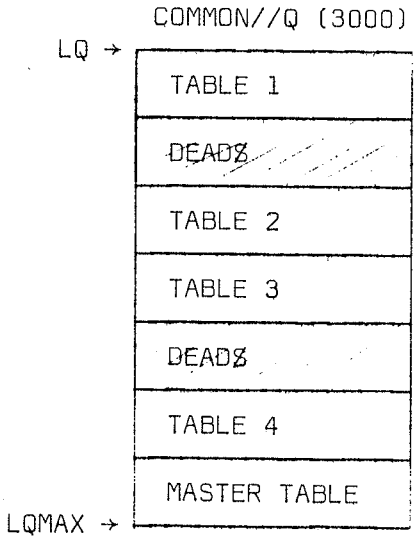


Figure 6

Every active table carries with it a unique 10 character Hollerith name whilst all dead tables are conventionally called DEAD$. (The $ appendage implies system produced tables.)

Each table is allocated in Q either with specified length or with an unknown length indication. In the latter case the system provides a default setting and the size is adjusted as execution proceeds. In this way the table sizes slowly adjust to the execution time requirements which are data dependent.

4.3. Control of Tables

All control information needed by TABMAN is held in the Master Table. Every table has a packed five word entry indicating its current status. Listed below are the items stored for each table:

i) Table Name (up to 10 characters)
ii) Length of Table
iii) Entry Length
iv) Location in Q
v) Page location in ECS
vi) Highest filled entry number
vii) Padding factor used for extension
viii) Frequency count (used by ECS swapping algorithm)
ix) ECS swop frequency (accounting)
x) Number of entries for dump request
xi) Disk unit number for disk rollin/rollout
xii) Disk Directory Table Number
xiii) Table Number of First record of RA file
xiv) Status Bits (13 currently defined - see below)

These are enough to describe the characteristics of the table. The status bits describe the current physical state of the table and control revolves around the setting of these bits, which are listed below.

Bit 15	Write Lock	
14	Extension forbidden	
13	Dead	
12	Roll-out to ECS permitted	
11	Rolled out to ECS	
10	Roll-out to Disk permitted	
9	Rolled out to disk	
8	Fixed Table	
7	Dump request	
6	In CM now	
5	Closed	
4	Converted according to collating sequence	} or sorting routines only
3	Sorted	} or sorting routines only

Most of these bits are self-explanatory but a few words might be added on some of them. A table is normally movable in Q but if bit 8 is set the table may not be moved. This is useful for tables which contain absolute pointers (e.g. I/O buffers, tables with pointers in). Write locking usually implies that the table has been created

in a previous run and is now intended as a "read-only" table. Schema tables would be typical examples of a use of this bit. Some tables may be movable but not extendable, bit 14 being used for this purpose. "Closure" implies that the table is probably complete and is reduced to its minimum necessary length. It does not preclude extension. Some of these bits may be manipulated by user subroutine calls as required. Some examples of these will be given after a discussion of user access to tables.

4.4. Accessing Tables

A table would be created by the following sequence:

```
NAME = 7LMYTABLE
CALL CRETAB(NAME,0,20)
```

Here one is creating a table called "MYTABLE" with an unknown length (indicated by the zero in the second argument) and with an entry length of 20 words. The system will then allocate a default table size at creation time. The name of the table is placed in the Master Table and the variable NAME is overwritten with the number of the table position allocated in the Master Table. The user does not normally need to know what this allocation is. All subsequent references to the table are usually through the variable NAME which is normally, but not necessarily, in COMMON. The table can similarly be removed from the system by a call CALL DESTAB(NAME). This results in the space occupied by NAME becoming a dead table with bit 13 set. Many of the status bits may be set using simple calls of this nature, e.g. CALL WLOCK(NAME) will set the write lock bit, or CALL FIXTAB(NAME) will declare the table to be fixed by setting the fixed bit. This automatically implies that no permission will be granted for roll-out of this table to ECS or disk.

Functions also exist for extracting the entry length and location of a table. e.g.

LOC	=	LOCTAB(NAME)	returns the address of the first word of the table NAME in Q
LN	=	LNTHE(NAME)	returns the entry length of table NAME
LAST	=	LASTE(NAME)	supplies the highest entry number so far placed in the table NAME.

Data may be transferred in and out of tables in the following way:

```
CALL PUTE(NAME,N,ENTVECT)
```

places an entry stored in the array ENTVECT into the Nth entry position of the table NAME, providing that the table is not write-locked. The entry number N may require space beyond the current size of the table. If it does, this will invoke the routine FINDSP to

increase the memory allocation to that table. Similarly

CALL GETE(NAME,N,ENTVECT)

will retrieve the same entry from the Nth entry position in the table and place it in the array ENTVECT.

Normally, all references to data in tables are made through these functions. However, in the case of fixed tables it is possible to perform direct transfers using the LOCTAB function to obtain the actual CM location of the table.

Each time a new table is referenced the Master Table entry must be unpacked (using COMPASS) into the COMMON block /QTABLE/. However successive references to the same table will avoid any repeated unpacking.

A complete list of all TABMAN routines and their functions is given in the Appendix.

4.4. Allocation of Tables and ECS Usage

Tables are normally allocated in CM if space is available. That is to say a search is first made for a dead area which is large enough to accommodate the CRETAB request. However if space is not available then central memory will be collapsed (by the routine MOVTAB) in order to coalesce the various potential DEAD$\emptyset$ tables. This process involves the relocation of all pointers in the Master Table, and may also free entries in that table at the same time.

If space is still not available then an algorithm is invoked (FINDSP) to choose tables which should be rolled out to ECS. Before discussing the algorithm in outline let us take a look at the ECS structure. ECS has been paged for convenience into 1000_8 word pages (this is a system parameter.) A page table (ECSMAP$\emptyset$) is held in CM in a fixed position with a one word entry describing the status of each page in ECS (fig. 7) i.e. occupied and chained or empty. Note that CM has not been paged to avoid overloading the retrieval routine GETE too much.

Each table carries an access frequency (updated by PUTE, MATCHE and GETE) which is used by FINDSP to determine which table is to be rolled out. The table with the lowest access frequency is chosen, and the ECS map is searched for empty slots. The table is then rolled out a page at a time into each empty slot (not necessarily contiguous). The page map chains together the pages associated with a single table. This process is often initiated by CRETAB or PUTE calls but may also be initiated by other routines, and is hidden from the user.

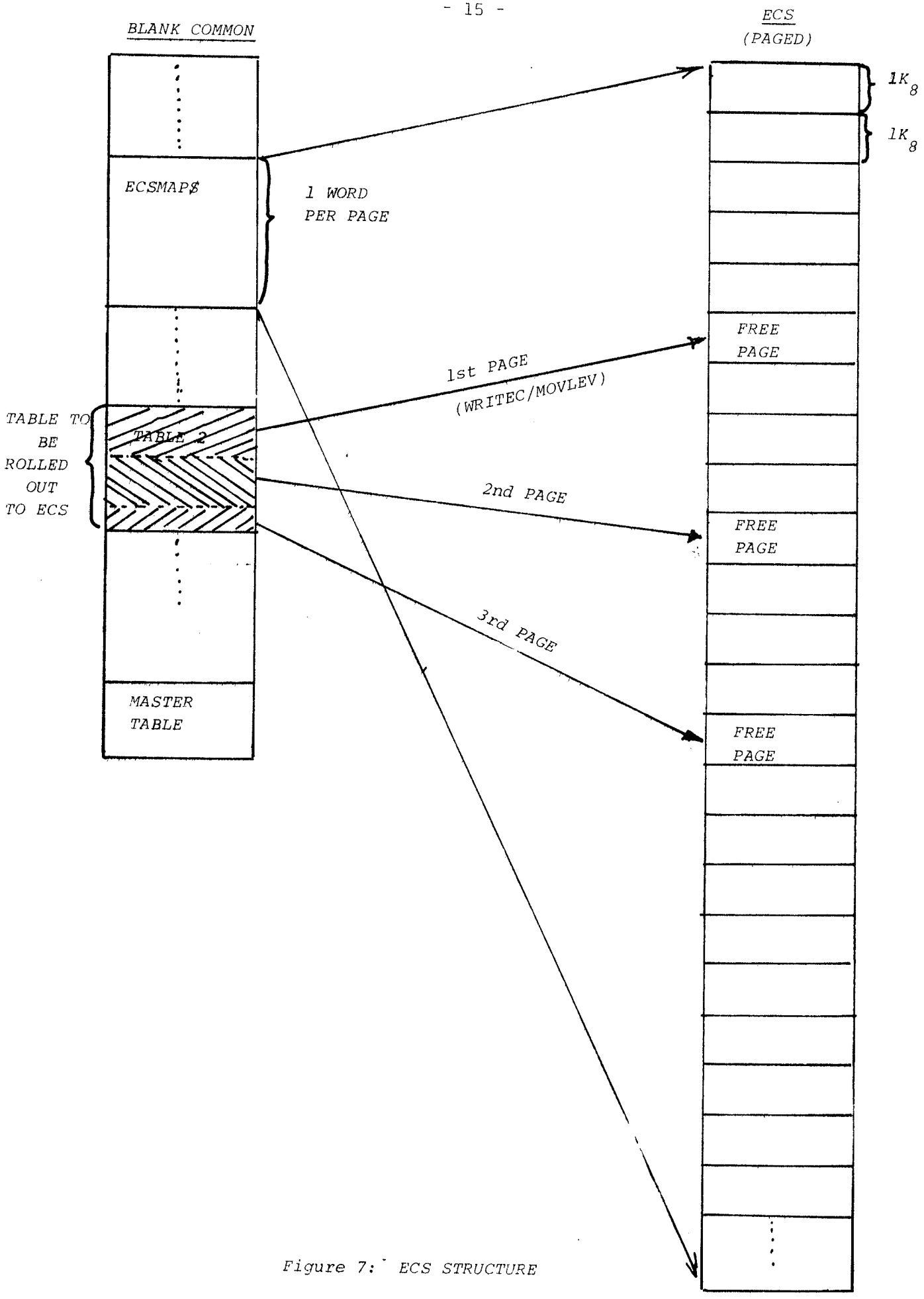


Figure 7: ECS STRUCTURE

Blank COMMON may or may not be collapsed in this process. If FINDSP is invoked as it may be by PUTE requesting more space than the table originally contained, the algorithm will attempt to satisfy the request first by extending the table if possible. However if this is not possible, as is often the case, then the entire table may be copied to a new larger dead area. If this is not possible, then a memory collapse and/or rolling out of selected tables will be tried until the request is satisfied. In a situation where CM is heavily loaded and there are many tables of "unknown final length" this shuffling process may be quite extensive. Efficiency is always gained if tables are created with approximately correct sizes.

If PUTE requests more space for a table the amount of additional space requested depends upon the "padding factor" associated with each table. This is preset to 10% but may be altered for any given table by the routine SETPADD. Space may also be requested in more precise, discrete units, by the routine REQSP(NAME,N) where N specifies the number of new entries required.

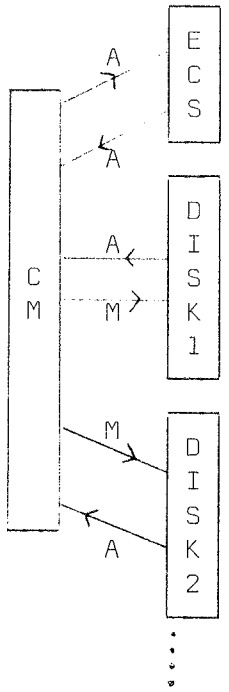
If a table entry is required by GETE or other routines from a table which is rolled out then the table will automatically be rolled in. It is possible with the routine GETECS to extract a single entry directly from ECS. This is provided for large, rarely used tables (e.g. error messages).

Should the ECS page map become full (i.e. ECS is full) then in principle the system could select tables to be rolled out to disk. However, although the hooks are present for this, it has not been implemented since

a) all ECS tables would have to be staged through a CM buffer to disk

b) it is considered better for the application to control the disk usage more closely than a general system could do.

The disk structure will be described after a discussion of the file handling routines. At this stage the storage levels and the degree of automation is shown in Fig. 8.



M : Manual or semi-automatic

A : Automatic

Figure 8: Storage Hierarchy

4.5. Search Techniques Using Tables

It is very common in the semi-systems type of programming required by TABLOID to search tables for entries which have specific characteristics, e.g. a particular bit set or a particular string of characters in a certain word. This was the main motivation for introducing the mask entry in the zero'th position. Four searching functions are provided

MATCHE(NAME,I,ENTVECT)	MATCHEC(NAME,I,ENTVECT)
MATCHZ(NAME,I)	MATCHST(NAME,NAME2)

MATCHE performs a linear search on table NAME from entry I to find the first entry which, when masked,is equal to the entry stored in ENTVECT. In this way we may search for an equal bit, character string or word. MATCHZ is similar except that it searches, for a complete zero entry in the table. Either function will cause the table to be rolled in if necessary. (A "Roll-in" of a table implies that the entire table is rolled in not just the appropriate page.)

The masks at the head of each table are initialised by CRETAB to be all ones, i.e. the entire entry is masked. The masks may

be reset at any time by the user calling the SETMASK or ADDMASK routines.

Both MATCHE and MATCHZ require that the table to be searched should be resident in CM. However one may wish to keep large infrequently accessed tables in ECS. To avoid rolling in these large tables two further search functions are provided.

MATCHEC is exactly equivalent to MATCHE except that the table must reside in ECS. A linear search is made to find an entry equal (under the mask) to ENTVECT. A scratch table is set up to store the mask and current entry during MATCHEC, and the scratch table is destroyed on exit.

MATCHST will perform a folded binary search on a sorted table stored in ECS. This is particularly useful for searching dictionaries. The second argument is the table number of a table specifying the mask and the type of the words in NAME. (See Appendix). MATCHST will also work if the table is in CM.

4.6. File Handling

One of the principal objectives of TABMAN is the ability to be able to open and close files of any type in a flexible manner. The allowable file types are sequential (SQ), random access (RA) (or word addressable) and Scope Indexed Sequential (SIS).

4.6.1. File Organisation

a) Sequential (SQ)

This is the familiar form that is produced by FORTRAN coded or binary WRITE statements. A file consists of records in the physical order in which they were written. No logical order exists other than their relative physical record position.

b) Word Addressable (RA) (or "Random Access")

A word addressable file is a mass storage file of records accessed by the number of the first word in each record. The file can be considered simply as a number of logically contiguous words, any of which may be retrieved by its word number. Each file has logical word addresses 1,2,....N up to the limit of a single volume (disk pack).

FORTRAN Extended (FTN)[3] contains a set of routines OPENMS, READMS and WRITMS, which impose a simple structure upon the basic file organisation These routines set up and maintain a single index record to the file

containing a list of all records in the file. This index list may name each record with a single 7-character key or the records may be accessed by number (i.e. numerical position in the index). TABMAN uses the number option for all random access files. Records stored in such files may be of variable length.

If an existing record is re-written with a new length exceeding its previous length then a complete new contiguous copy is written at the end of information.

This type of file organisation is well suited to a comparatively small number of records of variable length which are to be retrieved and updated randomly.

c) Scope Indexed Sequential (SIS) Files

A SIS file is a mass storage file of records stored in logically sequential order that can be accessed either randomly by key or sequentially by position. A standard CDC package called SIS (version 1) under SCOPE 3.3, (IS version 2 under 3.4) exists which creates and maintains such files[4].

A "key" must be supplied as an identifier with each record written to the file. Normally each record has a unique key but it is possible for duplicate keys to exist as an option. This key may be either an integer, a character string, or a floating point number. As records are written to the file SIS creates an index for the file. Both index and data records are maintained in sequence by ascending numeric key values or by ascending collating sequence in the case of symbolic character keys. SIS can locate any given record by searching the index record(s) for either the whole key or part of the key.

A SIS file consists of three parts:

i) Data Blocks which each contain a group of records and an ordered list of keys for those records.

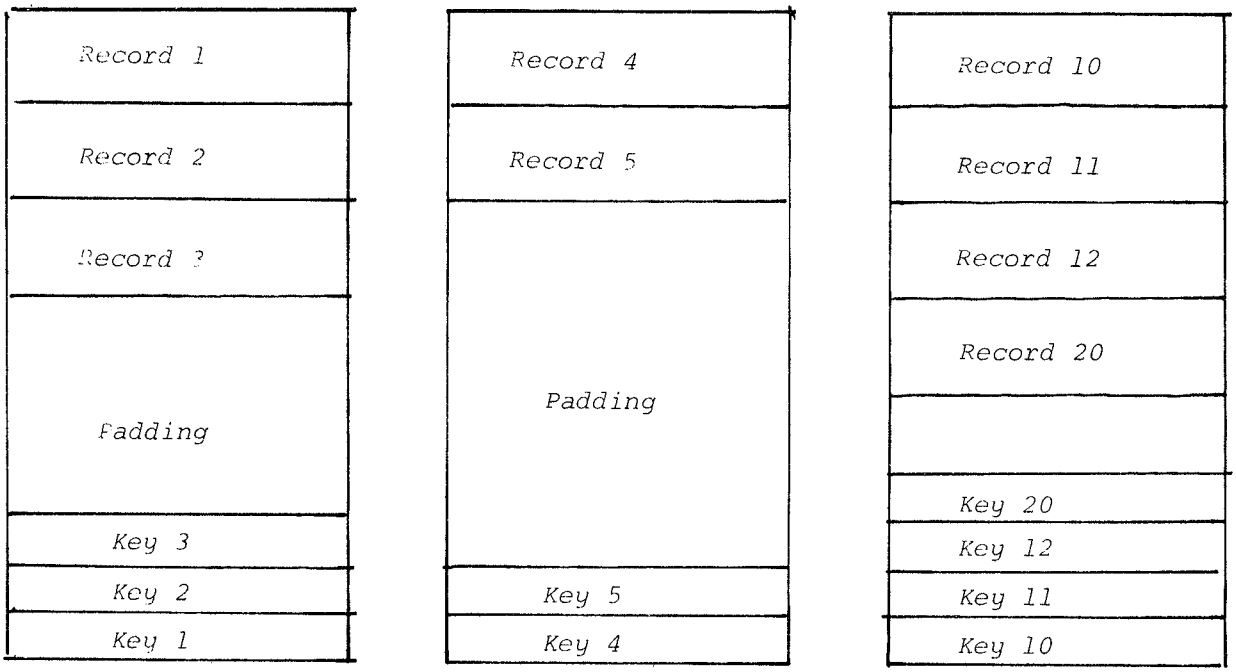
ii) Index Blocks containing a group of keys, each of which corresponds to the first record in a data block.

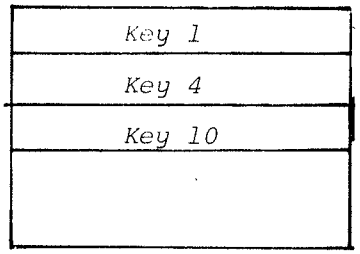
iii) A File Statistics Table (FSTT) which is created and maintained by SIS. This contains such information as the total number of records inserted or deleted, total number of accesses, etc.

The user defines the sizes of the data and index blocks at creation time but does not directly manipulate the information inside the blocks.

Both the key entries (at the bottom of the block) and the data records are stored in ascending order (see Fig. 9). When a data block becomes full, SIS will automatically split the data block into two, re-organising the records and keys appropriately; at the same

Data Blocks contain data records and keys





Index block keeps *entry points to data block containing keys with values from this key to the next key identified in index block*

Fig. 9: Indexed Sequential File with 1 Level of Index Block

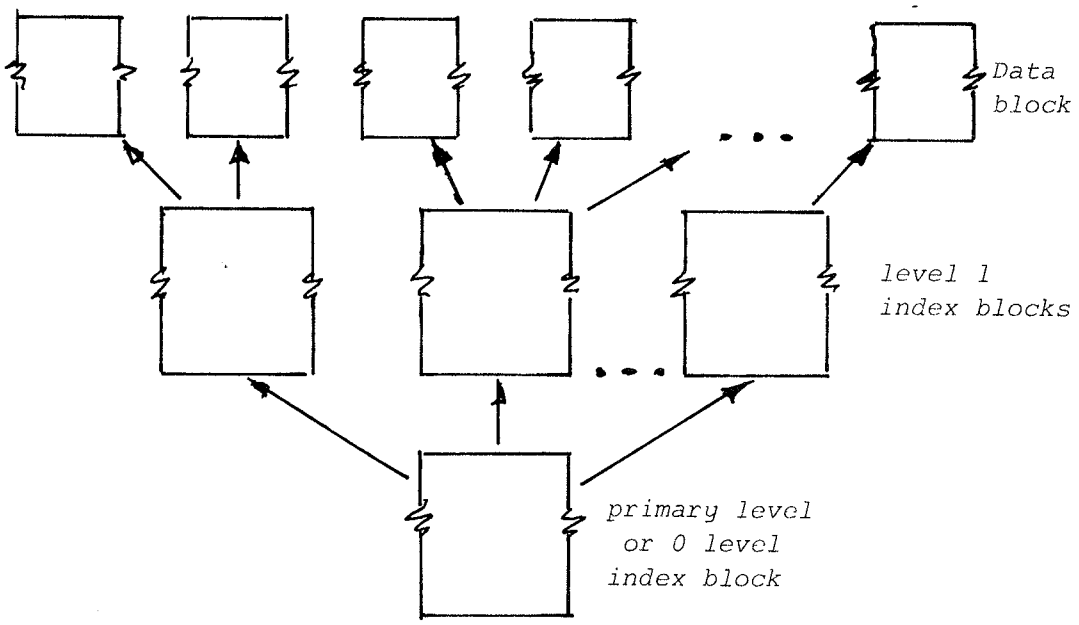


Fig. 10: Indexed Sequential File with 2 levels of Index Block

time a new entry is made in the index block for the new data block. Similarly if an index block becomes full it will automatically be split. This gives rise to a second level of index block (see Fig. 10). Up to 63 such levels are permitted.

The number of index levels is related directly to the time required to retrieve a record from the file. Each index block that has to be searched adds time. Adding records to a file does not necessarily increase the number of index levels if the amount of padding defined for both data and index blocks suffices for the additions that are to be made. However it may be that after long periods (months) of updating, deletion, etc., it may well be advantageous to re-structure the file to reduce the retrieval time.

Thus it can be seen that the file is always logically (but not necessarily physically) sequential and it can always be read in sequential mode by SIS routines. SIS can also retrieve on "major keys" which are defined as the leading left most characters in a symbolic key.

SIS data records may be variable length but TABLOID uses only fixed length records in order to maintain efficiency. Each variable length record is written as a number of fixed length records using the least significant character in the key to indicate the presence of multiple records representing a single logical record.

SIS file organisation is well suited to comparatively large files which can be organised around a single (possibly concatenated) key. For problems involving true multi-key access it is necessary to either have inverted files or to use an extension to SIS called SISTER [5] which has been developed by Temple University. Future versions of TABLOID may well make use of this extension.

4.6.2. File Access

Two basic routines are provided for handling files:

i) QIFILE(FILNAM,UNIT,DISP,TYPE,PARAM)

which will open a file called FILNAM, and assign it a UNIT number. Its disposition (DISP) may be 'LOCAL', 'ATTACH' or 'CATALOG' and of TYPE 'SQ', 'RA' or 'SIS'. PARAM is a vector specifying the file characteristics, e.g. buffer length or SIS parameters (most parameters of this type are defaulted by TABMAN so that if the user does not specify them the default values will be substituted). The function of QIFILE is to create the File Environment Table (FET) required by SCOPE 3.3 together with an associated I/O buffer as a fixed table in blank COMMON. Under SCOPE 3.4 the File Information Table (FIT) also has to be set up.

If the file is a random access one then additional tables are required for the index buffer requested in the OPENMS call, and for the RA file structure imposed by TABMAN. The FET (+FIT) and buffer table is called "TAPEmn" where mn is the allocated unit number.

If the disposition is 'ATTACH' then the appropriate permanent file macro is called to attach the permanent file to the program. For the disposition 'CATALOG' the file is first 'requested' to be on a Permanent File (PF) device and at program termination time, or earlier, the file will be catalogued using the macro facility.

ii) QDFILE(FILNAM)

closes the file FILNAM and then destroys all associated fixed tables thus releasing the space back to the Table Manager. If the file was to be catalogued the appropriate permanent file macro will be called just after the file is closed.

TABMAN maintains two internal tables to control file handling; a File Name Table called FILES$ and a File Control Table called FCT$. The FILES$ table is basically a copy of the execution list of files and their FET addresses which the SCOPE operating system requires in RA+2 onwards. This copy is maintained for ease of access by TABMAN. Whenever a new file is created or an old one destroyed, FILES$ is updated and then copied into RA+2 onwards.

The FCT$ contains all the information necessary to either attach or open a file. That is to say each file has an entry in the FCT$ containing local file name, permanent file name (if any), filetype, passwords, disposition, cycle number and parameters for creating the file, such as buffer length, etc. Entries can be made into the FCT$ without the file being active. A routine, QSFILE(FILNAM) will open a file which already has an entry planted in the FCT$. (QIFILE makes an entry in the FCT$ table, and then initialises it at the same time.)

Certain files are automatically initialised by TABMAN for use by TABLOID. These are OUTPUT, INPUT, MESSAGE, COMMAND, DATA, REPORT and DEBUG. Of these MESSAGE, REPORT and DEBUG are automatically equivalenced to the OUTPUT file using the routine QEFILE (file1,file2). Similarly COMMAND and DATA are initially equivalenced to the INPUT file. Unit numbers are assigned by TABMAN to INPUT and OUTPUT but the user does not need to know what assignments have been made.

A routine exists called QUFILE(file1,file2) for de-coupling equivalenced files. Files may be either local or permanent in the system thus enabling scratch files to be set up for sorting for example.

A file may be removed totally from TABMAN by calling QKFILE(FILNAM). This removes the FCT$ entry and RETURN's the file to SCOPE, thus making the file completely unknown outside TABMAN.

A typical example of TABMAN coding using some of these routines may be seen in Fig. 11.

```
C--
      DIMENSION IA(8),IB(4)
C--     OPEN A LOCAL SEQUENTIAL FILE CALLED MYFILE , BUFFER SIZE=65 WORDS
      CALL QIFILE(6LMYFILE,NUNIT,5LLOCAL,2LSQ,65)
C--     SET UP A TABLE TO STORE INFORMATION (UNKNOWN LENGTH)
C--     TABLE HAS 4 WORD ENTRIES
      NAME=7LMYTABLE
      CALL CRETAB(NAME,0,4)
      I=1
C--          READ FROM MYFILE
   10 READ(NUNIT,100) IA
  100 FORMAT(8A10)
C--     TEST FOR E.O.F.
      IF(UNIT(NUNIT).GE.0) GO TO 20
C--     SET UP ENTRY VECTOR
      IB(1)=IA(1)
      IB(2)=IA(3)
      IB(3)=IA(4)
      IB(4)=IA(6)
C--     PUT ENTRY INTO TABLE
      CALL PUTE(NAME,I,IB)
      I=I+1
      GO TO 10
C--     END OF FILE, CLOSE DOWN FILE
   20 CALL QDFILE(6LMYFILE)
C--     CLOSE DOWN TABLE, WILL RETURN SPACE TO TABLE MANAGER
      CALL CLOSTAB(NAME)
C--     RESET MASK TO REFER TO FIRST FOUR CHARACTERS OF SECOND WORD
      CALL SETMASK(NAME,11,14)
C--     SET UP SEARCH CONDITION
      IB(2)=4LCERN
C--     NOW SEARCH UNTIL FIRST ENTRY WITH WORD CERN APPEARS IN SECOND WORD
      I=MATCHE(NAME,1,IB)
C--     IF NO MATCH QUIT
      IF(I.EQ.0) STOP 77
C--     NOW ROLLOUT TABLE AND DROP THE CM VERSION
      CALL ROLLOUT(NAME)
      CALL DROPTAB(NAME)
```

Figure 11: Example of Programming using TABMAN routines

4.7. Random Access File Facilities

The way in which RA files are handled by TABMAN will now be explained. The access mechanism is 'word addressable' provided by the READMS/WRITMS function provided in FTN. All random access files in the system have the structure shown in Fig. 12.

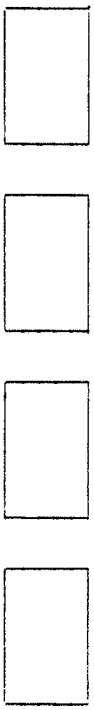


Figure 12

The records are written to the disk using the number option in the OPENMS call. The first record is used by TABLOID for book-keeping information and has a fixed length of 50 words. The second record is a directory of all the records on the file. Each subsequent record is a complete table and the directory entries are essentially just copies of the corresponding Master Table entries when the table is in core, e.g. table name, length and so-on. The directory is updated each time a table is rolled out to a disk file by ROLDOUT(NAME,FILE). The file created is completely self contained and may be catalogued after creation for use in a subsequent run.

When an existing RA file is opened with a QSFILE, the tables to hold the first two records are created and then read down. The directory entries are then transferred to the Master Table so that all the tables in this file are immediately known to TABMAN as rolled out tables. The allocated table numbers can be obtained through the function N=NTABLE(7LMYTABLE) for example. Any reference to these tables by GETE, MATCHE, etc. will automatically result in the tables being brought into CM. The Master Table entry contains all the relevant information to carry out these operations.

TABMAN also maintains a scratch RA file known as 'DSKROLL' as a file onto which users can temporarily roll out tables. Again TABMAN will automatically roll in any such tables that are referenced, but no disk roll out's are ever performed without a user request. In this way the scratch RA file can be considered as a back-up to ECS if the application needs a large number of tables.

The schema tables mentioned in the introduction to TABLOID are typical examples of tables which can be kept on a permanent RA file. The first record of the RA file is used by TABLOID to associate these schema tables with a specific SIS file.

4.8. SIS File Facilities

The SIS parameters for each file are stored in the FCT$ table. These define such items as data block size, index block size, padding factors, key size and so on. A default set of SIS parameters are defined in TABMAN so that the user need only specify those that he wishes to change. The standard CDC interface to SIS requires the user to specify the FET (or FIT) address. Since the user is normally unaware of the FET (or FIT) addresses which were allocated by the QIFILE or QSFILE calls, small interface routines have been provided so that the SIS files may be referenced by name. A COMMON block /SISBUF/ contains the current key and record. A call to

```
CALL RPSISF(FILNAM)
```

for example will replace the record in SIS file FILNAM having the key specified in /SISBUF/ with the new record which was also placed in /SISBUF/. Any error condition must be detected by inspecting SISST flag in /SISBUF/ after each call. The complete list of routines is given in the Appendix.

The SIS error message file SISMESS is regrettably outside the control of TABMAN since SIS performs its own FET initialisation without reference to the RA+2 area.

4.9. Sorting Facilities

Two levels of sort routines have been provided. At the first level any individual table may be sorted using the routine SORTAB. This is called as follows:

```
CALL SORTAB(NAME,KNAME,MVNAME,UP)
```

where KNAME is a table (which may optionally be created by SORTAB) which holds sorted indices to the table NAME on exit. The table has one word entries for each entry in NAME. MVNAME is a table which has only one entry of the same length as NAME specifying the types of the words in NAME. This enables mixed entries of integers, characters and floating point numbers to be sorted. The mask in MVNAME is used to mask the main table during the SORT. UP is either +1 or -1 indicating a request for either an ascending or a descending sort.

The sorting technique employed is based on the QUICKERSORT method[6] which itself exists as a CERN library routine (M 107). All characters in the original table are converted according to a collating sequence which is stored in /COLLATE/ prior to the sort. The user is responsible for re-converting the table with CONVTAB after the sort is complete.

At the next level a disk sorting routine (SORTFIL) is provided which takes as input an intermediate file which is blocked and can be considered logically as a large table (with the masks removed). The philosophy adopted is to first roll out all tables which can be rolled out in Q, and then to create as large a table as can be accomodated. This table is then filled, sorted and written to a new intermediate scratch file. The process is repeated, but this time merging the sorted table with the previously sorted information onto a second scratch file. This sort-merge process using two scratch files continues until the sort is complete. The scratch files are created and destroyed inside SORTFIL so that their lifetime is only for the duration of the sort.

4.10. Overlays

CERN is currently still using SCOPE 3.3 which implies that we must use Overlay Loading rather than Segment Loading (as in 3.4). However to improve the efficiency a scheme has been developed as a part of TABMAN to allow the primary and secondary overlays to reside in ECS and to be called directly from there, rather than be involved in disk accesses for each overlay. In addition the overlays are not re-read if the same overlay is re-entered on successive calls.

The scheme works in the following way. TABMAN (and TABLOID) are normally compiled with a blank COMMON large enough to accomodate the essential tables needed to initialize TABMAN; these are buffers for INPUT, OUTPUT and the tables FILES$ and FCT$. In addition a buffer is needed to read the overlay file, and a special table called OVERLAY$ is required to hold a 2 word entry for each overlay. This contains information giving the load address, entry point and length of the overlay.

The overlays are generated onto a file called TABLOID. The program is loaded and (0,0) is entered. After TABMAN has been initialised by QINIT the core image of blank COMMON appears as shown in Fig. 13. All the available dead space following the OVERLAY$ table is set up as a table called OVFIX$ which is fixed. A second initialisation routine QINIOV then reads down all the overlays except (0,0) from the file TABLOID (using BUFFERIN) and places them, paged, into ECS. At the same time an entry is made for each overlay in OVERLAY$. The OVFIX$ area must be large enough to hold the longest primary and secondary overlay. After all overlays have been read QINIOV is now able to release space (shaded) at the head of OVFIX$ and at the bottom since the length of the longest overlay is known. All primary overlays start at Q(N) where N is defined at loading time.

Thus the overlays are called down by a special routine called SUPRLAY(m,n) (where m,n are the overlay numbers), instead of the normal overlay call. This has the effect of rolling in the necessary overlay from ECS into Q and the code is executed within the table called OVFIX$. Normal TABMAN tables appear both above and below this fixed table containing executable code!

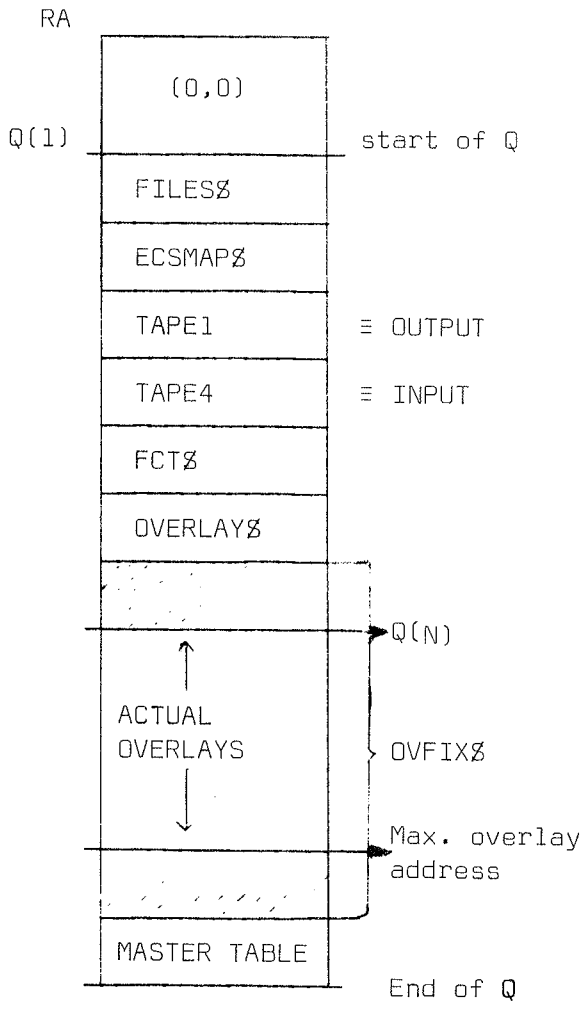


Figure 13

4.11. Overall Structure of TABMAN

TABMAN is maintained as an UPDATE file which contains a number of different options. Binary libraries of the various versions are kept as permanent files on the CDC 6400 (using CROVER[7] under SCOPE 3.3 and LIBEDIT under SCOPE 3.4).

The three principal options are selected by defining the following flags:

*DEFINE DEBUG	Debug versions which includes checks on the arguments of most of the principal routines. All Compass routines are excluded.
*DEFINE MESSAGE	Inserts messages into many routines so that a trace of TABMAN actions may be obtained. This results in a sequence of messages such as:

```
TABLE  NAME1  CREATED              (CRETAB)
TABLE  NAME1  FIXED                (FIXTAB)
TABLE  NAME2  ROLLED OUT TO ECS    (ROLLOUT)
TABLE  NAME3  DESTROYED            (DESTAB)
  :
  :
etc.
```

*DEFINE ACCOUNT — Includes accounting information which can be examined by the routine QSTAT. Provides for example the total number of tables created and destroyed, statistics on ECS, and disk usage.

If none of these options are selected then TABMAN will be compiled into its most optimised form which includes COMPASS versions of MATCHE, GETE, PUTE and UNPACK.

If the messages are compiled with the *DEFINE MESSAGE option the actual messages may still be suppressed by setting the logical variable MESSAG (in /QCNTRL/) to .FALSE.

4.12. Use of TABMAN

Since TABMAN performs its own I/O buffer initialisation the binary libraries contain two COMPASS versions of the main program called MAIN and MAINOV. These correspond to the non-overlay and overlay versions of TABMAN respectively.

The user of TABMAN must provide a subroutine which is effectively his main program called MAINP and is called automatically from these COMPASS main programs after TABMAN has been initialised. To make clear the functions of the two actual main programs the flow in the two cases is shown below.

MAIN		MAINOV
Call RECOVR →	for mode 1 recovery	← Call RECOVR
Call XSETIO →	for I/O error processing	← Call XSETIO
Call QINIT →	to initialise TABMAN	← Call QINIT
	to initialise overlays on ECS	← Call QINIOV
Call QINIERR →	to initialise error messages on ECS	
Call MAINP →	User main program	← Call MAINP
Call QFINISH →	Close down all files and STOP	← Call QFINISH

4.13. Error Handling

All I/O errors on sequential files are currently detected by the UNIT function. (However, if tape files are to be used these should probably be converted to X-package calls[8].) All SIS I/O errors are handled automatically by SIS (or by 6RM in SCOPE 3.4).

If a mode 1 error (address out of range) occurs the job is "reprieved" and TABMAN traps to the error recovery routine QRECOVR which prints out all relevant TABMAN COMMON blocks together with the Master Table and a short dump of each table currently in the system.

All errors inside TABMAN itself are fatal and all error handling is performed by the routine EPRINT. This routine has a single argument n which is an error number in the range $1 \leq n \leq 38$. On encountering an error EPRINT is called with a unique number and a message will be placed in the dayfile. If the MESSAGE option has been selected then more detailed error messages are stored in a table (ERRTAB$) which is kept on ECS (routine QINERR). In this case a corresponding explanatory message will appear in the OUTPUT file. After the error message is printed all files are closed down by QFINISH and program terminates with a STOP66.

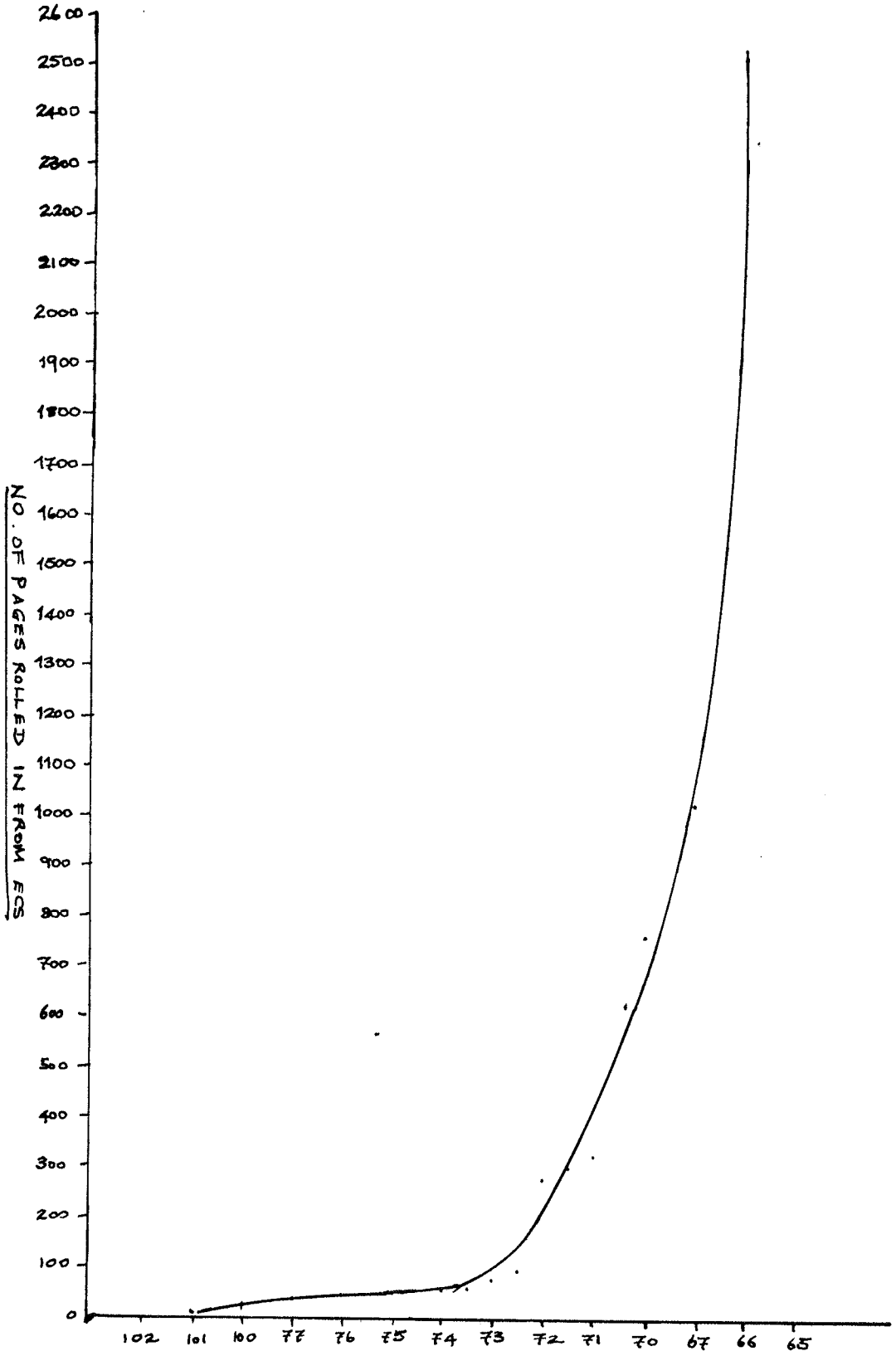
The DEBUG version contains about twice as many calls to EPRINT as the non DEBUG version.

4.14. Performance

Fig.14 shows how the number of rollin/rollouts increases as the amount of CM space is slowly decreased for the same job. Beyond a certain point TABMAN clearly exhibits the expected "thrashing" phenomenon in which the same tables are constantly rolled in and out. The upper portion of the curve is not completely smooth since the size of the larger tables tend to dominate the rollin/rollout. The lower portion of the curve also indicates that for this job there is only a small time penalty to be paid in reducing the CM space from $102k_8$ to $73k_8$. There is probably an optimum point for any job of this nature where the CM space can be traded for only a nominal increase in CP time used.

4.15. Documentation

A measure of self-documentation has been used for TABMAN and TABLOID and has been provided by using conventions for comment cards which were defined at CERN for HYDRA programs. This enables a program called WRUP2 to pass over the FTN output listings to produce the type of output seen in Fig. 15. While this is not considered sufficient documentation it provides at least a quick guide to the functions of each routine and the meaning of the arguments which must be supplied. The output can be cut up since it is A4 size and filed



CORE ASSIGNED (1000_8 WORDS)

FIG. 14 ECS PERFORMANCE

T A B M A N V E R S I O N 4

PAGE 46

```
**********
*        *
*  MOVTAB  *
*        *
**********
```

FTN

```
SUBROUTINE MOVTAB(LS,LE)
************************

     COMPRESSES TABLES IN BLANK COMMON

-------------------------------------------------------------------------
I                                                                       I
 GENERAL REMARKS                                                        I
I                                                                       I
I LS,LE ARE OPTIONAL PARAMETERS SPECIFYING LIMITS OF PARTIAL COLLAPSE   I
I IF NOT GIVEN WILL COLLAPSE WHOLE OF BLANK COMMON                      I
I                                                                       I
-------------------------------------------------------------------------

COMMENTS
--------
    NFREE GIVES SPACE FREE IN LAST TABLE ABOVE MASTER TABLE
    NFREE GIVES MAXIMUM CONTIGUOUS SPACE FREE
    FIND TABLE IN MASTER TABLE
    SHOULD ALWAYS FIND MATCH
    RELOCATE LOC
    DEAD TABLE FOUND
    INCREMENT RELOCATION COUNT
    CLEAR SCM POINTER
    CLEAR ENTRY IF NOT ROLLED OUT
    CONTINUE UNTIL MASTER TABLE REACHED
    SET LSCM TO END OF LAST GOOD TABLE
    ENTER DEAD TABLE AT END
    SCAN FOR HIGHEST NON-ZERO TABLE NUMBER
    RESET NTAB TO HIGHEST NON-ZERO ENTRY

EXTERNALS
---------
ENDT                                    EPRINT
MOVE                                    NOARG
UZERO

COMMON BLOCKS
-------------
/ /
QTABLE
QCNTRL
QBITS
QMASK
QACCNT
```

Figure 15

into a book. The write-up program produces an index to the routines which can be placed at the start.

5. Conclusions

The TABMAN package has proved to be of considerable help in developing the TABLOID retrieval program. It is now essentially complete although studies could be made to find improved algorithms for rolling out tables and perhaps to make more automatic use of disk tables. The conversion to SCOPE 3.4 will be carried out shortly. The additional features of the Record Manager in Scope 3.4 would in fact make it more attractive to consider a completely paged version of TABMAN (i.e. CM, ECS and disk).

As TABLOID applications develop it may also be worthwhile to consider incorporating the Temple University extensions to SIS (called SISTER) into TABMAN.

6. Acknowledgements

TABMAN owes much to previous dynamic core storage systems developed at CERN for the bubble chamber programs LBCG and HYDRA and also to an earlier table manager developed by B. Lautrup for compiler writing.

I would also like to acknowledge the many helpful discussions I have had with my colleagues who are developing TABLOID, particularly with M. Clayton, W.G. Moorhead and S. Santiago.

7. References

1. B. Sagnell, Cable and Equipment Information Retrieval Programs. Lab. II-CO/INT/GE/MEMO/BS/hk/72-11, October 1972.

2. Data Base Task Group, April 1971 Report. CODASYL Committee (available from ACM).

3. FORTRAN Extended Reference Manual, CDC Publication No. 60305600B.

4. Scope Indexed Sequential Reference Manual, CDC Publication No. 60305400D.

5. SISTER: Scope Indexed Sequential Temple Extended Retrieval Package. External Reference Specifications. Systems Software Group, Temple University, January 1972.

6. QUICKERSORT, Algorithm 271, Comm. ACM 8 (Nov. 65) p. 669.

7. CROVER, Extended Version CERN Program Library K420. Allows private libraries under SCOPE 3.3.

8. X-Package. CERN Program Library Long Write-up Z200.

APPENDIX A

Routine Description

A. User Routines (F) = Function

A.1 Table Manipulation

(NAME = Table Number unless otherwise stated)

1. ADDBMSK(NAME,IB1,IB2)

   Add bits to existing binary mask in table NAME

   IB1 = Bit position 1
   IB2 = Bit position 2

2. ADDMASK(NAME,ICH1,ICH2)

   Add characters to existing character mask

   ICH1 = Character position 1
   ICH2 = Character position 2

3. CHTYPE(CHAR) (F)

   Supplies type of character

   CHAR = Character in left most 6 bits

   CHTYPE = -1 alphabetic
   0 numeric
   +1 special character

4. CLOSTAB(NAME)

   Close Table (i.e. release unwanted space)

5. CLRTAB(NAME,IWORD)

   Clear Table

   IWORD = value table is to be cleard to (e.g. zero)

6. COLTAB(NAME,IWORD)

   Collapse Table

   IWORD = value of entries which are to be ommitted. Whole entry must be same, under the mask, as IWORD.

7. CRETAB(NAME,N,LN)

   Create Table

   NAME = Table Name on entry: Table Number in exit
   N = Number of entries
   LN = Entry length

8. CUTTAB(NAME)

   Cuts table to current length and padding

9. DESTAB(NAME)

   Destroys Table

10. DROPDSK(NAME)

    Drop the random access version of table.

11. DROPECS(NAME)

    Drop the ECS version of table.

12. DROPTAB(NAME)

    Drop the CM version of table.

13. DUMPTAB

    Dump all tables having the dump bit set in their status words.

14. EXTEND(NAME)

    Inhibit extension of table

15. FIXTAB(NAME)

    Fix Table (i.e. inhibit movement)

16. GETE(NAME,I,VECT)

Get entry I from Table NAME and place in array VECT.

17. GETECS(NAME,I,VECT)

Get entry I from Table NAME direct from ECS if possible, and place in array VECT.

18. LASTE(NAME) (F)

LASTE = Highest entry number in table name.

19. LNTHE(NAME) (F)

LNTHE = Entry length of table NAME.

20. LOCTAB(NAME) (F)

LOGTAB = CM location of table NAME.

21. MOVTAB

Collapse blank COMMON (2 arguments for system use only, are not obligatory).

22. NTABLE(NAME) (F)

Returns table number of table NAME (Hollerith).

23. OPENTAB(NAME,N,LN)

"Open" a table that exists on an RA file. Arguments are same as CRETAB. Table is not rolled in. That is: entry in Master Table for rolled out table.

24. PUTE(NAME,I,VECT)

Put entry stored in VECT into I'th entry position in table NAME.

25. QROLL(L,N)

Roll out all possible tables to ECS.

L = location of largest free space
N = length of largest free space.

26. QSTAT

Print current statistics

27. QTDUMP(NAME)

Dump table NAME according to mode set in Master table entry by SETDUMP.

28. REQSP(NAME,N)

Request more space for table. If N is zero request amount specified by padding. If N is non zero: request this number of new entries.

29. ROLDIN(NAME)

Roll in table from disk.

30. ROLDOUT(NAME,FILNAM)

Roll out table to disk. FILNAM is an optional second argument file name to be rolled to. If omitted "DSKROLL" is assumed.

31. ROLLIN(NAME)

Roll in table from ECS.

32. ROLLOUT(NAME)

Roll out table to ECS.

33. SENDDSK

Select ECS tables to be rolled out to disk. Not coded, presently dummy.

34. SETBMSK(NAME,IB1,IB2)

Set up binary mask for table from bit position IB1 to bit position IB2 (from left to right).

35. SETDUMP(NAME,NWD,ITYP)

Request dump of NWD entries of table name in mode ITYP = 1 characters
2 fl. point
4 Integer
8 Octal
e.g. ITYP = 10 = fl. pt and octal.

36. SETENT(NAME,N) — Reset number of entries in table NAME to be N.

37. SETMASK(NAME,ICH1,ICH2) — Set up character mask for table NAME from character position ICH1 to character position ICH2. (from left to right)

38. SETPADD(NAME,IPADD) — Re-set padding factor in table (preset to 10 = 10%).

39. SETROLD(NAME,I) — Set roll out to disk flag in table to allowed (I=1) or dis-allowed (I=0)

40. SETROLE(NAME,I) — Set roll-out to ECS flag to allowed (I=1) disallowed (I=0).

41. UNFIXT(NAME) — Un-fix table.

42. WLOCK(NAME) — Write-lock table (i.e. inhibit writing by PUTE).

A.2 Search Functions

All functions return either entry number satisfying the condition or else zero if condition was not met.

1. MATCHE(NAME,I,VECT) — Linear search for an entry equal to entry stored in VECT starting from the I'th entry. Each entry in table is masked using the table mask. The table must be in CM (if not it will be rolled in).

2. MATCHEC(NAME,I,VECT) — Same as MATCHE except that the search will be carried out by calling down the entries from ECS one at a time if table is only in ECS. No rollin performed.

3. MATCHST(NAME,NAME2) — Binary search of table stored in ECS or CM. If in ECS the table will not be rolled in. NAME2 is table number of an auxiliary table which specifies the mask to be used and the type of each significant word in the entry. The table NAME must be sorted by the key which is searched for.

4. MATCHZ(NAME,I) — Linear search of table NAME starting from entry I to find the first zero entry. Table will be rolled in if necessary.

A.3 File Manipulation Routines (excluding SIS)

1. QIFILE(FILNAM,NUN,DISP,TYPE,PARAM) — Enters File FILNAM (up to 7 characters left justified, zero filled) into FCT% and then initialises the file by setting up the FIT and FET buffer in Q and then opening the file.
NUN is allocated unit number.
DISP is disposition: may either be 'LOCAL' 'ATTACH' or 'CATALOG'. If of type 'ATTACH' the appropriate p.f. will be attached. If of type 'CATALOG' file will be catalogued at closure.

TYPE = SQ = sequential
RA = random access (word addressable)
SIS = Scope Indexed Sequential

PARAM are file characteristics
SQ : PARAM(1) = buffer length

RA : PARAM(1) = buffer length
PARAM(2) = number of expected records (maximum)

SIS PARAM(14) the fourteen SIS parameters (see more detailed write up).

2. QSFILE(FILNAM,NUN) — Initialises file already entered into the FCTØ.

3. QEFILE(FILE1,FILE2) — Equivalence file1 to file2.

4. QUFILE(FILE1,FILE2) — Uncouple FILE1 and FILE2 which were previously equivalenced.

5. QDFILE(FILNAM) — Close file FILNAM, destroying all associated tables. Will 'catalog' or 'extend' file as necessary.

6. QKFILE(FILNAM) — Close file FILNAM, destroying all associated tables and 'return' file to SCOPE. Will catalog or extend file as necessary.

7. LUNIT(FILNAM) (F) — Returns unit number associated with file FILNAM.

8. SQREAD(IN,SPROUT) — Reads from sequential file on unit IN into ISBUF in 8A10. The record is then blown out into JSQBUF(80). If IN≤0 then no physical read is performed but input buffer is set by the external routine SPROUT.

9. SQWRITE(IOUT) — First bunches JDATA(136) into IDATA(14) and then writes record onto sequential file on unit IOUT. If IOUT=0 no physical writing takes place.

A.4 Sorting Routines

1. SORTAB(NAME,KSNAME,MVNAME,UP) — Sorts table NAME. Produces an index list in table KSNAME which may optionally be created before SORTAB is called. This index table must contain space for a one word entry for each entry in the table to be sorted. MVTABLE is a further table which has the same entry length as NAME and has only a single entry which specifies the type of each word in the table. (1 = character, 2 = floating point, 4 = integer, 8 = octal). The mask in MVNAME is used to mask the main table during the sort. For character entries the sorting is performed according to the collating sequence stored in the COMMON block /COLLATE/. UP = ±1 indicates either ascending or descending sort.

Disk Sorting is performed using the routine SORTFIL. The intermediate file to be input to SORTFIL is a blocked file produced by the routine WIFILE. This blocking is designed to give fast binary reading and writing and since each block containing an integral number of records is set up to be as close as possible to a single physical record unit (PRU) of 64 words on the CDC 844 disk packs for economy of storage.

2. WINIT(NUN,LEN) — Initialise buffer for writing intermediate file. NUN is unit number and LEN is entry length of file to be sorted. WINIT creates an intermediate table called WITAB$\emptyset$ which has the block size needed for the file.

3. WIFILE(NUN,IND) — Places a record which is currently stored at Q(IND) into the output buffer WITAB$\emptyset$. When the buffer is full it is output on unit NUN.

4. WILAST(NUN,ISW) — Writes out last, possibly incomplete buffer. If ISW=0 then WITAB$\emptyset$ is destroyed.

5. RINIT(NUN,LEN) — Initialises buffer for reading intermediate file. NUN is unit number and LEN is entry length of file. Creates a table called RITAB$\emptyset$ as the intermediate buffer and reads in the first buffer full.

6. RIFILE(NUN,IND) — Places current record into Q(IND) onwards. When buffer becomes empty it is automatically re-filled.

7. RILAST(NUN) — Destroys RITAB$\emptyset$ after all reading has been completed.

8. SORTFIL(FILE1,FILE2,MVTAB,NREC,UP) — Sorts FILE1 onto FILE2 using all of the above routines. MVTAB is the table needed by SORTAB to specify mask and type of words in FILE1.
NREC is total number of records in file.
UP = +1 for ascending sort
UP = -1 for descending sort.

9. CONVTAB(NAME,KAR,MVTAB) — Converts all characters in table NAME according to the collating sequence stored in the vector KAR. MVTAB is the table specifying the type of each word in the table.

A.5 Overlay Routines

1. SUPRLAY(M,N) — Calls down overlay (m,n) which has been stored on ECS. Is exactly equivalent to the normal OVERLAY call except that the file name is ommitted.

A.6 SIS Routines

1. INISISS(FILNAM) — Open SIS file FILNAM for sequential processing.

2.	INISISR(FILNAM)	Open SIS file FILNAM for random processing
3.	OPSISF(FILNAM,N)	Set up and open SIS file FILNAM according to disposition in FCTØ table; i.e. either as a new file or as an existing file.
4.	INSISF(FILNAM,N)	Insert record stored in COMMON block /SISBUF/, into SIS file FILNAM. N is record length if different from default. Key is also stored in /SISBUF/.
5.	RPSISF(FILNAM,N)	Replace record with key in KEY by the record in /SISBUF/. N is the record length if different from default value.
6.	DLSISF(FILNAM)	Delete record whose KEY is in /SISBUF/.
7.	CLSISF(FILNAM)	Close SIS file FILNAM.
8.	FCSISF(FILNAM)	Forced write-out of last buffer of SIS file.
9.	SRSISF(FILNAM,N)	Reposition file by ±N records.
10.	RDSISF(FILNAM,N,MKEY)	Read record with major key length MKEY from file into /SISBUF/. N is the record length found.
11.	NXSISF(FILNAM,N,MKEY)	Read next record with major key length MKEY into /SISBUF/. N is the retrieved record length.
12.	SLSISF(FILNAM,N,MKEY)	Seek data block associated with major key length MKEY. Does not read the record-only positions for computation overlays.
13.	SKSISF(FILNAM,N,MKEY)	Seek next block (index or data block with major key length MKEY. Positions file only for computation overlays. (This routine will be suppressed under SCOPE 3.4).

8.1 System Routines (not normally called by users)

1.	DEPWORD(IADR,WORD)	Deposits WORD into the address IADR.
2.	ENTDT(NW,LL)	Enters dead table stored at CM address LL, of length NW words into Master Table.
3.	ENTEROV(NTRYPT)	Enters overlay at NTRYPT which has been read down by SUPRLAY.
4.	ENTMT	Ensures that Master Table has at least 2 free entries. May invoke table collapse to achieve this.
5.	ENTNT(NAME)	Enters a new table into Master Table.
6.	EPRINT(N)	Called upon detection of fatal error in TABMAN. N is the error number.
7.	EXTMT	Extends Master Table if possible. Called by CRETAB when number of active tables becomes critically near to the length of Master Table.

8. FINDSP(NAME,LENT,LTT)

Finds space for table NAME. This may either be the complete space or an extension of an existing table. In either case the routine may invoke the algorithm for rolling out tables. LTT is the position of free space in CM.

9. FLIST

Copies FILES$ table into RA+2 area to maintain up-to-date list of files and their FET(FIT) addresses for communication with SCOPE.

10. INPUTN. (or NAMIN. m3.4)

Provided to allow TABMAN to have its own private NAMELIST reading routine which allows character input (which the FTN version does not).

11. IRTNAD(IADR) (F)

Fetches the SUPRLAY return address for current overlay from the stack.

12. ISISFET(FILNAM) (F)

Fetches FET addresses associated with SIS file name FILNAM and sets up corresponding FCT$ entry in /QENT/.

13. MAIN and MAINOV

Main programs (see main text).

14. NAMIN(IN,NAMLIST)

IN is unit number.NAMLIST is namelist vector created by FTN. Performs namelist type input - allows character input instead of floating point.

15. QINIERR

Initialises error messages stored in ERRTAB$ onto ECS. Should be called by user in first overlay.

16. QINIT

Initialises blank COMMON and various variables of TABMAN.

17. QRECOVR

Error recovery routine which dumps COMMON blocks after MODEl error. After dump calls QFINISH and exits.

18. QINIOV

Initialises overlays onto ECS.

19. SENDDSK

Dummy routine which is meant to select ECS tables to be rolled out to disk.

20. SETFCT(IN)

Initialises FCT$ table from Namelist input.on unit IN Used only for stand-alone tests of TABMAN. Replaced by more elaborate routine inside TABLOID.

21. SETRTN(IADR)

Set up return address for SUPRLAY on the stack that is provided.

22. UNPACK(NAME)

Unpacks entry for table NAME into COMMON block /QTABLE/.