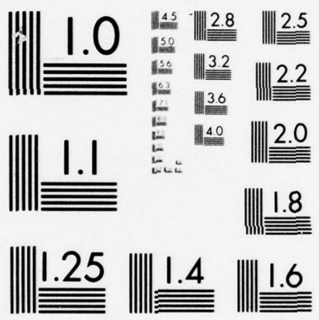
AD-A056 203 NATIONAL MILITARY COMMAND SYSTEM SUPPORT CENTER WASH--ETC F/G 9/2
NATIONAL MILITARY COMMAND SYSTEM INFORMATION PROCESSING SYSTEM --ETC(U)
JUN 76

UNCLASSIFIED NMCSSC-TR-80-72-CHANGE-1 NL

| OF |
AD
A056203

END
DATE
FILMED
8 -78
DDC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

DEFENSE COMMUNICATIONS AGENCY
COMMAND AND CONTROL
TECHNICAL CENTER
WASHINGTON, D. C. 20301

6 C 8

LEVEL III

(6) National Military Command System Information Processing System 360 Formatted File System (NIPS 360 FFS). NIPS Processing Handbook. Change 1.

IN REPLY REFER TO:

(11) 10 June 1976

TO: DISTRIBUTION

-AD-766135

(12) 65p.

SUBJECT: Change 1 to TR 80-72, NIPS Processing Handbook, dated 1 February 1973

(14) NMCSSC-TR-80-72-CHANGE-1

1. Insert the enclosed change pages and destroy the replaced pages according to applicable security regulations.

2. A list of Effective Pages to verify the accuracy of this handbook is enclosed. This list should be inserted before the title page.

3. When this change has been posted, make an entry in the Record of Changes on the inside cover.

FOR THE DIRECTOR:

[illegible]

68 Enclosures
Change 1 pages

J. DOUGLAS POTTER
Assistant to the Director
for Administration

243 000

JCB

EFFECTIVE PAGES - 10 JUNE 1976

This list is used to verify the accuracy of TR 80-72 after change pages have been inserted. Original pages are indicated by the letter O, change 1 by the numeral 1.

Page No.	Change No.
TITLE	O
ii	O
iii-x	1
1-22	O
23-24.2	1
25-26	O
27-36.1	1
37-38	O
39-42.1	1
43-128	O
129-130	1
131-132	O
133-134.2	1
135-188	O
189-190	1
191-200	O
201-215.2	1

CONTENTS

Section Page

ILLUSTRATIONS

TABLES

ABSTRACT

This Technical Report identifies and documents characteristics of the many factors affecting NIPS performance which should be considered when defining a NIPS file and coding NIPS procedures. This is a supplement to and not a substitute for the following NIPS user documentation:

Volumes I-IX;

CSM UM 15B-60--	Vol I	- Introduction to File Concepts
	Vol II	- File Structuring (FS)
	Vol III	- File Maintenance (FM)
	Vol IV	- Retrieval and Sort Processor (RASP)
	Vol V	- Output Processor (OP)
	Vol VI	- Terminal Processing (TP)
	Vol VII	- Utility Support (UT)
	Vol VIII	- Job Preparation Manual
	Vol IX	- Error Codes
CSM GD 15A-68--		General Description; and
TR 54A-70 --		Installation of NIPS 360 FFS.

1. Percent of file updated
2. ISAM records in overflow
3. Change or add data
4. ISAM pad records

c. Retrieval
   1. Percent of file searched
   2. ISAM records in overflow

d. Terminal Processing Requirements

e. File Backup Requirements

3.2.1.1 File Generation

File generation normally is more efficient for a SAM file than for an ISAM file. Each SAM file record can be written in sequence without the ISAM processing overhead, such as maintenance of cylinder and track indexes and generation of pad records.

3.2.1.2 File Update

The percent of the file updated is a major factor in determining the relative efficiency of updating SAM and ISAM files. If a relatively small portion of the file is to be updated, ISAM generally will be more efficient than SAM, since only those records changed or added are accessed. As the percentage of the file updated increases, the relative advantage of ISAM over SAM decreases, until the point is reached where SAM processing becomes more efficient.

The number of records written to the overflow area affects the efficiency of ISAM processing, and hence, its efficiency when compared with SAM processing. Changes to ISAM file fields will normally not cause overflow records, since these changes replace existing data. The effect of adding new data depends on the location of the data in the file. If data is added between existing logical records in the file, ISAM processing normally causes either the new or existing logical records to be written to the overflow area. If new records are added to the end of an ISAM file, generation of overflow records depends on the record IDs of the pad records at the end of the file. Pad records are generated at the end of an ISAM file with keys greater than the highest record created, and these keys are incremented based on the record IDs that were created. Only six pad records, blocked at 1000 bytes, will fit on a 2314 track. Therefore, if all new records within the range of the high and low keys to be placed on each pad record track do not fit, some of the new records must be written to the overflow area.

Occasionally, a programmer generates an ISAM file with only a small portion (e.g., 10 percent) of the data records in order to thoroughly review the results prior to adding the remainder of the data. Performing an ISAM update to add the remaining data (e.g., 90 percent) to the file will be very inefficient, and possibly terminate because of lack of space. The probability of the new records being forced to the overflow area is increased, since the algorithm developing the records IDs of the pad records was based on a small sampling (initial generation) of the total data records. The run terminates if the data written to the overflow area exceeds the overflow space allocated.

3.2.1.3 Retrieval

Two techniques exist in RASP and QUIP which can limit the portion of the file which must be searched. They are:

a. LIMIT

b. SECONDARY INDEXING.

Both techniques permit the retrieval component to access only a range of records or specific records based on record IDs. Retrieval from an ISAM file will normally be more efficient if LIMIT or secondary indexing can be used. These techniques will be less effective for a SAM file, since each data record must be read up through the upper limit of the LIMIT range, or the highest record ID for secondary indexing candidate records.

When a batch output job is required to access an entire NIPS file (no LIMIT or indexing is used), SAM file processing is usually faster than ISAM. In one test, the execution time required by RASP and OP to process an ISAM file with overflow records was 2 hours and 17 minutes. After the file was reorganized and the overflow records were placed in the prime data area, the ISAM processing time was reduced to 35 minutes. When a SAM version of the file was processed by the same RASP and OP job steps, the execution time was reduced to less than 13 minutes. A twelvefold decrease run time between processing an unorganized ISAM file and a SAM file was observed in this test. Similar results have been shown in other tests. The processing of ISAM files containing a significant number of records in the overflow area will adversely affect the processing efficiency of both RASP and QUIP. Thus, before starting any extensive batch output cycle from an ISAM file, the file should be reorganized.

3.2.2 Inquiry Processing

Normally, response requirements are imposed on the information retrieval operations against a file. The file must be designed to satisfy these requirements.

Response requirements may take two forms--time requirements and format requirements. Stringent time requirements may dictate an on-line data base, while less rigid requirements may leave the designer free to choose. Similarly, complex format requirements may dictate the use of the more general and powerful output processor, while a simpler format could be satisfied by a QUIP-published product. The requirement for a complex report in a very short time could present operational problems if special considerations are not made.

Two methods are available for processing inquiries against NIPS data files: QUIP, the on-line (or batch) retrieval/output component, and RASP/OP, the more powerful data retrieval and output tandem that only operates in the batch mode. Although both perform similar functions, they satisfy different requirements. QUIP satisfies on-line retrieval and output requirements and provides a quick and simple batch capability. RASP/OP is a powerful capability that permits more complex processing than does QUIP.

In the batch environment, the user may select which method to use. QUIP employs interpretive processing techniques and performs its function in a single job step. RASP/OP is a two step operation and both components employ generative processing techniques. As a result QUIP often requires less run time to accomplish a given task, especially when a relatively small number of records is queried. When possible, both methods should be tried to determine which is more efficient under the given conditions.

Both RASP and OP may be made more efficient through the utilization of stored process elements and LIMIT/OMIT logic. Stored process elements are those queries and RITs that have been previously compiled and stored on an execution library. They may be called in and executed immediately, thus avoiding the otherwise routine and time-consuming step of compilation every time they are required. Use of LIMIT/OMIT logic permits the associated component to bypass those file records that cannot meet requirements. If used wisely, this capability can reduce the number of records processed.

Organization of data to make effective use of the SELECT operator in RASP, can lead to significant savings in I/O time, QDF data set space requirements, and subsequent processing time.

3.2.3 Secondary Indexing

Secondary indexing is a special processing capability developed to increase processing efficiency for certain types of files, and should be considered during file design.

Secondary indexing is a process of specifying additional retrieval control fields for a file to enhance information retrieval. In this case, however, uniqueness of the index value is not a requirement, as many records probably have the same index value. Thus, for information retrieval, a file's primary index is the record control group while other data fields may be specified as secondary, or alternate, indexes.

As a means of comparison, consider the RASP LIMIT statement. Through its use, examination of records may be limited by the contents of the high-order positions of the record control field. If secondary indexing is utilized, examination of records for retrieval may also be minimized, but based on the content of the secondary index, which may be <u>any</u> fixed field value in a fixed or periodic set, or any word in any field in a fixed, periodic, or variable set (keyword index).

The benefit resulting from the correct use of secondary indexes is that of dramatically reduced retrieval - run time. For an ISAM data file with secondary indexing specified, only those data records qualified by the secondary index are accessed; these selected records are then subjected to the complete logic of the query. The majority of the file's records may be ignored.

For a SAM data file, secondary indexing is not quite as significant, since the file must be passed sequentially simply to access the records qualified by the index. In this environment, however, retrieval terminates when the last qualifying record has been processed. Also, for large multi-volume SAM files, secondary indexing will specify which volumes are to be searched.

Consider the example shown in Figure 7. The user has specified that the field LOC is an index field. The Index Data Set, TEST360X, contains all record IDs for each unique value (CANNES, PARIS) of each index field (LOC). When the user executes the query:

```
IF LOC EQ PARIS
```

the Index Processing phase of RASP or QUIP evaluates the query and determines (from the index descriptor record in the FFT) that LOC is an index field. The Index Processing phase determines from the Index Data Set the record IDs (J00042, J00123, W04237, W05689 and W06210) of each record in the file which has a LOC value of PARIS. These records now become candidates for evaluation by the retrieval component. In this example, all candidates, in fact, satisfy the query. If, however, the query had been specified:

```
IF LOC EQ PARIS AND MEQPT EQ HU-15.
```

not all candidates would satisfy the query. Since MEQPT is not an index field, the retrieval component must evaluate each of the candidate records (LOC EQ PARIS), and qualify only those records which also have a MEQPT value of HU-15.

Secondary indexing is not without its expenses. Secondary indexes must be defined, either during file structuring for new files or by the index specification utility (UTNDXSPC) for existing files. Space for these indexes must also be reserved in the form of the Index Data Set. Index Data Set space requirements may be computed using the following algorithm:

# 1. For each indexed field, compute the number of blocks (BR) required to store the unique values from that field:

$$\frac{(VL+7)*NUV}{BLOCKSIZE} = BR$$

User selects Index Fields

Index LOC.

Index Data Set contains:

Record ID's for each value of each Index Field

Evaluate only Candidate Records

Query: IF LOC EQ PARIS.

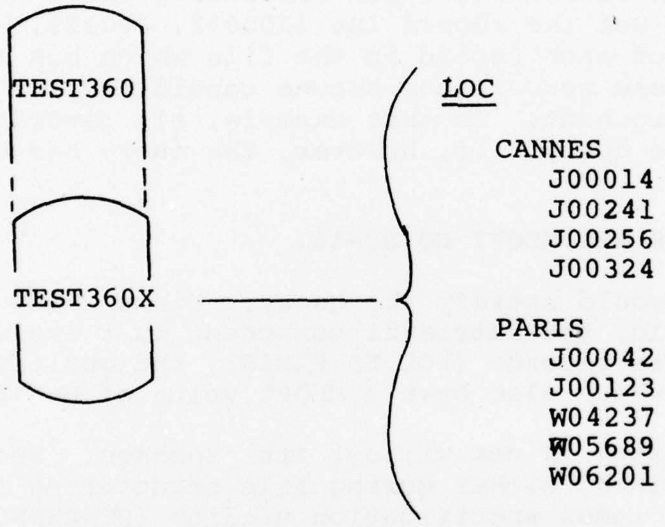


Figure 7. Index Data Set

2. For each indexed field, compute the number of blocks (MR) required to store master indexes:

$$\frac{(VL+7)*BR}{BLOCKSIZE} = MR$$

3. For each unique value in each field, compute the number of blocks (CR) required to store the major identifiers of the records in which the value appears.

$$\frac{NR*(RL+2)}{BLOCKSIZE} = CR$$

4. Compute the total block requirement by summing **the counts for items** 1, 2, and 3.

$$BR + MR + CR = NBR$$

DEFINITIONS:

#	NUV -	Number of Unique Values or Keywords
#	VL -	Value Length or Average Keyword Length
	RL -	Major Record ID Length
#	NR -	Number of Records (or Subset Records if the Index is a Periodic Field/Group) in which a unique value or keyword appears.
	CR -	Candidate Requirement
	BR -	Base Requirement
	MR -	Master Requirement
	NBR -	Number of Blocks Required (if Gen Mode, Allocate 1.15 X NBR).

Effective use of this algorithm, however, requires that the user have an exceptional knowledge of the data values in his file. If the user is not able to accurately estimate his data set space requirements, he can use the following procedure:

a. Allocate a large amount of space on a disk pack.

b. Generate the Index Data Set using File Maintenance or the UTNDXSPC utility program. At the end of this run the message:

```
**3599** OF THE XXXXX RELATIVE BLOCKS ALLOCATED
TO THE INDEX DATA SET, YYYYY ARE USED.
```

will be printed. This message will tell the user the exact number of blocks required for the Index Data Set.

c. Use the XTRDISK procedure for the UTNDXTFR utility program to transfer the Index Data Set to tape.

d. Scratch the Index Data Set from the disk pack.

e. Use the XTRTAPE procedure for the UTNDXTFR utility program to transfer the Index Data Set to disk. Allocate more blocks for the Index Data Set than the current requirement (indicated by the message from file maintenance (FM) or UTNDXSPC) to allow for Index Data set expansion. The Keyword Analysis Utility (UTNDXKAN; XKA procedure) can be used to obtain keyword data for use with this algorithm.

The Index Data Set will automatically be updated (index maintenance) by FM each time the data file is updated, provided the XINDEX symbolic parameter references the Index Data Set. The relative cost of index maintenance depends on the frequency and volume of index field updates. Addition of new records containing index fields will always require updating of the Index Data Set. Change transactions affect the Index Data Set only if the index fields are affected.

The Index Data Set is maintained as a BDAM file. As this data set is updated, its organization gradually becomes less efficient. It is necessary to periodically reorganize the Index Data Set by using the UTNDXTFR utility program to transfer from disk to tape and then back to disk. Each time the Index Data Set is updated the number of blocks utilized is printed. The gradual increase over the original requirement gives an indication of the need for reorganization.

Since the Index Data Set must be disk resident, the requirement for backup must be considered. A procedure similar to the procedure recommended for ISAM file backup (subsection 3.2.1.5) could be implemented. By reorganizing (XTRDISK, XTRTAPE) the Index Data Set after each update, the user not only ensures optimum organization of the Index Data Set, but also provides a backup Index Data Set. If the Index Data Set is destroyed or (through user or operator error) gets out of synchronization with the data file, the user must generate his Index Data Set in order to continue using the secondary indexing capabilities. To regenerate the Index Data Set, the user must take the following steps:

a. Erase the existing Index Data Set from disk.

b. Execute the UTNDXSPC utility program. Use the GEN option. The index specification utility program uses the index descriptor records already present in the data file FFT to generate a new Index Data Set.

Example:

```
//     EXEC    PGM=IEHPROGM
//DD1    UNIT=(2314),DISP=SHR,VOL=SER=123456
     SCRATCH   DSNAME=TEST360X,VOL=2314=123456
//    EXEC   XSP,XINDEX=TEST360,XDISP=NEW,
//    XVOL='SER=123456',ISAM=TEST360,
//    VISAM='SER=123456',PARM=GEN
```

In order to ensure the integrity of the indexing capability, it is mandatory that the Index Data Set correspond exactly with its associated data file. Each time an indexed data file is updated (by FM or SODA), a date of update is entered into the File Format Table (FFT) and the same date added to the Index Data Set. Whenever RASP or QUIP accesses the data file, the file date is compared to the Index Data Set date. These two dates must match in order for index processing to be invoked. If the dates do not match, the following message will be printed:

```
**3500**  INDEX USAGE NOT FEASIBLE.
          SEARCHING THE ENTIRE FILE.
```

In this case, the retrieval will still be properly executed, but indexing will not be used.

When FM or SODA accesses the data file for update, the data file and Index Data Sets are also compared. If they do not match, the following message will be printed:

```
**3418**  INDEX DATA SET NAME DOES NOT MATCH DATA
          FILE
```

The data file will be satisfactorily updated, but the Index Data Set will not be modified. Therefore, since the Index Data Set is no longer compatible with the data file, index processing will not be invoked in subsequent retrieval runs. To use index processing again, it is necessary to scratch and regenerate the Index Data Set.

If the user (intentionally or inadvertently) updates his data set, but does not include the indexing parameter (e.g., XINDEX=TEST360) on his FM EXEC card, the data file will be updated, but the Index Data Set will no longer be valid. Omission of the indexing parameter from a RASP or QUIP run against an indexed file, causes the retrieval component to determine that indexing is not feasible, but the retrieval will be satisfactorily executed.

When using Keyword Indexing, no safeguards are provided when a user changes keyword tables, because the tables are independent of specific data files. One stop word table may be specified for a number of fields in several data files. When a keyword table is changed, the Index Data Sets for all fields associated with the table no longer correspond with their associated data files. The affected indexes must be recreated by deletion and addition or the affected Index Data Sets must be scratched and regenerated. During the period when the table has been changed, but the Index Data Sets have not been brought up to date, erroneous results will be obtained from all NIPS functions.

One method for controlling data file access during table maintenance is to rename the table just before it is updated. After the table is updated, delete all affected indexes and add them with the new table name. In the interval, all attempts to access the indexed fields will result in "table missing" errors. If this method is used, remember that each page of the table must be renamed.

Advantages of using secondary indexing also depends on several factors.

a. Percent of file retrieved

b. Number of unique values

c. Frequency of retrievals

d. Number of index clauses

e. Frequency of index value change

f. Necessity for querying full text data in variable fields and sets.

Indexing is relatively ineffective if a large percentage of the file is retrieved. If a large portion of the file is to be retrieved it is more effective to bypass index processing (PARM='INDEX=NO') and sequentially access the records, rather than access them as candidate records one record at a time. The INDEX=NO option also bypasses the index processing overhead cost.

The more unique values a field has, the greater its potential effectiveness as an index field for retrieval, though the maintenance costs can be expected to increase. For an indexed field the length of the list of records containing each unique value, relative to the number of records in the file, affects the efficiency of using that field. If the list for each value is of equal length, the ideal number of unique values is equal to the square root of the number of records in the file.

For example, assume that LOC and CNTRY are fixed fields in a 1000-record TEST360 file, where there are 100 unique values of LOC and 20 unique values of CNTRY. Conditioning on LOC as an index field would qualify an average of 10 candidate records, while conditioning on CNTRY as an index field would qualify an average of 50 candidate records.

If indexed retrievals are infrequently utilized, the costs of index maintenance may be more than the savings accrued by retrieval.

Generally, it is desirable to define index fields and design queries to produce an explicit candidate list which causes retrieval evaluation of a relatively small number of records. Defining index fields to be used in multiple clauses connected by an AND can define a candidate list where every record satisfies the query. Excessive index processing overhead can be generated, however, if there is a broad overlapping of candidate records qualified by each index clause. For example:

```
     IF LOC EQ PARIS AND CNTRY EQ FRANCE.
```

Assuming there are 10 records in PARIS and 50 records in FRANCE, index processing would generate a list of record IDs satisfying the LOC EQ PARIS clause and a second list of the record IDs satisfying the CNTRY EQ FRANCE clause. These two lists would then be merged to form the single candidate list of 10 records satisfying both clauses. This candidate list would then be evaluated by the retrieval component. It is clear that there is redundancy in index processing for the evaluation of these two clauses, since every PARIS record is in FRANCE. Even if there is a possibility of one or more units in PARIS in some other country, exclusion of a CNTRY as an index field would reduce the index processing overhead cost and probably improve overall query response. In the above query, if CNTRY were not an index field, the candidate list would contain all units in PARIS (but not necessarily FRANCE). The retrieval component would then qualify only those records in PARIS FRANCE. If both fields are indexed, the overhead associated with CNTRY can be avoided by use of the FURTHER statement. Example:

```
     IF LOC EQ PARIS.

     FURTHER CNTRY EQ FRANCE.
```

FURTHER statement clauses are not processed by Secondary Indexing.

Effective use of the secondary indexing capability requires a thorough knowledge of data values and file utilization. Ideally, index fields would have low update activity, but frequent retrieval utilization.

To aid in selecting index fields, the user should utilize the file utilization statistics capability (see subsection 3.5). File utilization statistics inform the user of the most frequently referenced fields in each of FM, RASP, QUIP and OP. Using these statistics, the user can select for indexing those fields most frequently used by RASP and least frequently updated while avoiding data fields rarely used for retrieval but frequently updated.

The user should also employ the Keyword Analysis Utility (UTNDXKAN) to determine if keyword indexing would be practical. One of its functions is to list all words in a field with frequency counts or major record identifiers.

#3.2.4 Keyword Indexing

Keyword Indexing is a text-retrieval capability that provides a method by which the NIPS user can access and retrieve records based upon the contents of variable-length or text data fields. The Keyword Indexing capability is described in section 3.7 of the NIPS User Documentation, Volume I - Introduction to File Concepts.

Efficient use of Keyword Indexing requires an evaluation of the user application and the intelligent selection among a number of user options. The criteria for evaluating these options is discussed in the following paragraphs.

#3.2.4.1 Scan Subroutine

A scan routine defines word boundaries (i.e., determines the limits of words by identifying literals and text words) in keyword indexed fields during execution of FM, SODA, and Index Specification. The user can use the system provided scan routine, or can develop a special purpose scan subroutine for his application.

The System Scan subroutine defines the following word types:

a. Literal word - delimited by a single quotation mark character

b. Alphanumeric - letters and numbers

c. Decimal notation - numbers with embedded commas and a single period

d. Symbols and abbreviations - alphanumeric words with imbedded periods and hyphens (optional).

In addition, it recognizes a hyphen optionally as a textual character, as part of a symbolic word, or as a connector between two parts of a word that was split in the process of creating data file update transaction records.

If the System Scan Subroutine does not meet a user's needs, he can write one of his own. His subroutine must conform to the following interface conventions.

A scan subroutine is called conventionally by the Scan Processor (IXFMKSC):

```
register 15 - scan entry address
register 14 - return address
register 13 - save area address
register  1 - parameter list address.
```

The parameter list is shown in figure 7.1. The first time the subroutine is called, bits in the return code byte indicate end-of-field (new field to be processed). The scan subroutine controls the return code. It must clear the code when it begins to process a field and must set the end-of-field code when the field has been processed. The Scan Processor tests the code to determine when a new field should be addressed; only then will it update the parameter list. The scan subroutine must, before it returns, either move a one-byte word length and a maximum 30-byte word to the area addressed by a parameter, or it must set the no-word and end-of-field return code. When a literal word is returned it must flag the word length byte.

The keyword index hyphen option is available as a scan flag. The parameter binary equivalents to the index option are:

1 -	DROP
2 -	RETAIN
3 -	SEPARATE
4 -	TEXT (default)

The option words can obtain any meaning desired by the user.

It is suggested that the user employ the Keyword Analysis Utility (UTNDXKAN) to debug this scan subroutine. He must store the routine in a user library. Then he can specify its name in a control statement and limit the number of NIPS records processed during the test.

```
F - address of field HOP
F - scan limit address (field LOP)
F - address of word holder area
     area format:
          CL1 - word length. Bit 0 is 1 if literal.
          CL30 - word, left justified, no padding.
CL1 - hyphen option; binary.
          1 - DROP
          2 - RETAIN
          3 - SEPARATE
          4 - TEXT (default)
CL1 - return code
          bits 0-3:  not used by scan
          bit  4:    no-word-found if 1
          bit  5:    end-of-field if 1
          bits 6-7:  not used by scan
```

Figure 7.1. Scan Subroutine Parameter List

#3.2.4.2 Stopword Tables

A stopword table causes the elimination of words which match the table during index maintenance. Its function is to reduce the volume of data to be processed.

The decision for creating stopword tables should be based entirely upon volume. All occurrences of all words from keyword indexed fields are processed twice. First, they are blocked in index transaction records for passage to the OS sort. The space required is roughly equivalent to that required to store them in the data file. Then they are sorted (except SODA). Each word is stored in a fixed length field (31 bytes) in a sort record. A stopword table should probably be created when the volume exceeds 50,000 words, although this figure will vary with hardware capacity.

The system stopword table is intended for use with full text data that consists of grammatically correct sentences. It will probably have little value with data composed only of keywords or phrases, but such fields normally will not require a stopword table.

Note that the suffixed word function is not associated with stop words. To eliminate the suffixed forms of a word, each suffixed form must be defined as a stop word.

#3.2.4.3 Dictionary Tables

A dictionary table causes the substitution of the root form of a word for all occurrences of suffixed forms of that word and the substitution of one word for all words in a group defined to be synonymous.

The decision for creating a dictionary should be based on data content. If no dictionary is specified for a field, each unique word which does not match a stopword table is stored in the Index Data Set with the major identifiers from the NIPS records in which it appears. Under this condition, suffixed forms of a word are not recognized as such; each suffixed form is stored as a unique value. A dictionary is required if all forms of a word are to be stored as one value in the Index Data Set. The same reasoning applies to words which are synonymous or which the user wishes to treat as equivalent.

The same logic also applies to retrieval. Even if data file information does not include suffixed words or synonyms, the presence of a dictionary permits a user to include suffixed forms or synonyms in retrieval statements.

A dictionary can be employed to perform the function of a stopword table - the elimination of words which will never be used as retrieval terms. Words which match a stopword table are excluded; words which do not match a dictionary are excluded. However, the use of a dictionary solely for word exclusion is inefficient since no processing time or space is saved. Words are matched with the stopword table immediately after they are recovered by the Scan subroutine. Words are matched with the dictionary during the creation or update of the Index Data Set. In addition, each retrieval statement keyword term is matched with the dictionary.

#3.2.4.4 Data File Creation and Update

Data content, often tolerable in any form when used only for display, becomes critical when the same data is queried. Normal data validation procedures should be followed. In addition, the following considerations are pertinent to keyword indexing.

If raw data is coded before it is converted to machine readable form, keyword indexing permits the codes to take the form of several words or phrases stored in a fixed or variable length field. Although requiring more space than symbolic codes, this form simplifies error checking and eliminates the need for display conversion. A glossary of coding terms applied here would ensure consistency and probably eliminate the necessity for keyword tables; in effect, the coder matches the keyword dictionary before the data enters the system.

When FM update transactions are created, a procedure should be followed which prevents word splitting. If a word is split between transaction records, the System Scan subroutine will treat each part as a separate word. If a word is split within a transaction record, the hyphen DROP option may be used to recover it. For example, if transaction records are created from MTST tapes and several lines are combined into one record, a word may be split between lines and the parts may be separated by blank characters. If the first part of the split word is immediately followed by a hyphen (common continuation) and the DROP option is specified, the Scan subroutine will ignore the hyphen and any following blanks and recover the two parts as one word. The choice of this option prevents the use of the hyphen in symbolic notation, however.

Care should also be exercised in the use of the quotation mark character (quote). A single quote is a literal delimiter; two single quotes separated by any other character are required to define the literal word limits. (Two consecutive quote characters are ignored.) If only one quote is present in a field, the System Scan subroutine assumes that a literal word has been split. However, unlike its treatment of split non-literal words, it disregards all characters from the single quote to the end of the field. If a literal word is split between two records, the part of the literal in the first record will be ignored. The second part of the literal will be processed as a non-literal, and all data from the delimiting quote in the second record to the end of the field (which probably includes non-literal words) will be disregarded. In other words, an even number of quotes must be present or erroneous scanning will result.

Spelling errors can usually be corrected more easily in transaction records than in data files. Therefore it is recommended that all transactions be displayed and proofread before data file update.

It is also recommended that the MVR operator (POOL) rather than MVF be used in logic statements that create and update variable sets. MVF reblocks transaction data into field length subset records with a high percentage of word splitting; each subset is scanned as an independent field. MVR creates a subset record for each transaction record; transaction data is not reblocked. MVR may also be used to update variable fields; processing is identical to that of MVF. The NFL MOVE (variable field) and ATTACH (variable set) both expand into MVR operators.

#3.2.4.5 Data File Analysis

The objectives of analysis are to determine if keyword indexing of selected fields is feasible, to identify misspelled and split words in prospective fields, and to establish the contents of keyword tables if any are required. They are accomplished by examining word lists produced by the Keyword Analysis utility (UTNDXKAN). The following steps are recommended:

-- Feasibility: list all words in selected fields
-- Error correction
-- Create stopword tables
-- Create dictionaries

o Feasibility

For each selected field, prepare a UTNDXKAN statement; specify the field name and bypass for both tables. If the estimated volume of words from all fields will exceed 150,000, an initial run with a file statement that limits processing to 100 NIPS records may be advisable. Use the OS sort delete and maximum counts to estimate total volume and adjust the UTNDXKAN sort space parameter accordingly. Remove the file statement before making the analysis run.

The judgment as to whether or not it is feasible to index a field is completely subjective. It is a matter of weighing retrieval requirements against data content. This initial listing shows a NIPS record frequency count with each word which will probably be most helpful in analyzing data consisting of full sentences such as abstracts.

o Error Correction

It should be relatively easy to spot broken and misspelled words. If there are relatively few such words they can be ignored, i.e., eliminated by stopword table entries, or corrected by dictionary synonym entries. If the data is in rather bad shape, it will probably pay to correct the data file before continuing with the analysis procedure. As an aid in the correction process, add the record identifier option to the field statements of fields with poor data and execute UTNDXKAN to get a list of words with NIPS major record identifiers and occurrence counts. It is possible that a relatively few NIPS records contain all the errors.

o Create Stopword Tables

A stopword table will probably not be needed for fields that contain only coded words or phrases unless it is used to eliminate erroneous words. It would be better to correct words with a dictionary than to eliminate them. However, if split words are corrected, one part of the split must be retained for correction while the other part must be deleted.

For full-sentence data, word frequency counts can be used as a guide to the selection of stop words; words which appear in more than 40 percent of the data file records are probably not worth retaining.

If several or all fields contain similar data it would be advisable to create one stopword table for all of them to reduce overhead. In that case, it might be helpful to execute UTNDXKAN to obtain one list of words from all fields by using a file statement that specifies the merge option.

Punch the stop words into cards. If a user library does not exist, create one; the Dictionary Maintenance utility (XKM) procedure does not include parameters for library creation.

Prepare a UTNDXKMD table statement and a display statement for each stopword table to be created. All words must fit one table page. About 115 words can be stored on a 1K page; specify page size according to word volume.

After UTNDXKMD is executed to create the stop tables, revise the UTNDXKMD field statements; replace the stop bypass parameters with the proper table names, then execute UTNDXKMD to get a word list with stop words flagged. Use this list to validate stopword table content.

o Create Dictionaries

If stopword tables were created, add a parameter to each UTNDXKMD field statement to bypass stopword display and execute UTNDXKMD to get a list of non-stopwords only. Examine the list to determine if a dictionary is required. A dictionary is probably required if error correction is necessary, if suffixed words appear in the data file, or if synonyms are desired.

One approach to compiling a dictionary is as follows. First, examine the list produced by UTNDXKAN for suffixed words. Cross off the suffixed words and write the root form of the word next to them on the listing if it does not appear in the data file. If the word ending changes when it is suffixed, include suffix notation with the root form of the word. Next, make a list of all synonyms and check them off the UTNDXKAN listing. Do not include suffix notation in this list. Do include error correction synonyms. Note that a word included in the dictionary need not appear in the data file. All occurrences of a word may be incorrectly spelled, or a preferred retrieval term may not appear at all. The correct or preferred word can and should be included in the dictionary. Next, make a list of suffixes -- all root words on the UTNDXKAN listing with suffix notation. The same word may appear on both the list of suffixes and the list of synonyms. Check off all words with suffix notation on the UTNDXKAN listing. All words on the UTNDXKAN listing which were not checked off constitute a list of keywords (neither suffixed nor synonyms).

Generally, it will not pay to combine terms from several fields into one dictionary. The saving in overhead during retrieval and maintenance is insignificant. On the contrary, large dictionaries mean many pages, degrading rather than enhancing retrieval time.

After dictionary terms have been selected, prepare four UTNDXKMD table statements for each dictionary to be created -- to add keywords, suffixes, and synonyms, and to display the created table. After executing UTNDXKMD, replace the dictionary bypass parameters in all UTNDXKAN field statements with the names of the proper dictionaries and execute UTNDXKAN. Check the resulting list for proper word identification and substitution.

#3.2.4.6 Indexing the Data File

When executing the Index Specification utility, only keyword table page sizes and word volume require special consideration. If the sum of table page sizes exceeds 10K, the execute region should be increased by the excess for more efficient processing. If any page exceeds 10K, the region must be increased.

Computing the number of blocks to be allocated to the index data set is tedious at best. As an alternative to the algorithm presented in section 3.2.3, an approximate count can be obtained by counting the number of keywords and basewords on the final analysis UTNDXKAN listing. By observation, obtain a rough approximate average frequency count for all keywords, basewords, and convert words. Compute the block allocation from the following algorithm. Ignore the remainder in all divisions.

(1) Compute words per block.

$$WPB = \frac{\text{BLOCK SIZE}}{15}$$

(2) Compute keys per block.

$$KPB = \frac{\text{BLOCK SIZE}}{\text{KEY LENGTH}}$$

(3) Compute blocks required for words.

$$BVL = \frac{NKB}{WPB} + 1$$

(4) Compute blocks required for a master index; if BVL is less than WPB, MIL=0.

$$MIL = \frac{BVL}{WPB} + 1$$

(5) Compute blocks required for keys.

CDL = (AVE/KPB + 1) * NKB

(6) Compute minimum blocks required.

TOTAL = 2 + BVL + MIL + CDL

#3.2.4.7 Maintenance Considerations

When a data file is updated, its associated Index Data Set is automatically updated also. Keyword indexing affects this function only insofar as volume is concerned. Each keyword field contains multiple values (words); it will obviously take more time to update a keyword field than it will to update a secondary index field. In addition, Index Data Set reorganization will probably be required more frequently.

When a keyword table is updated, Index Data Sets associated with it are not automatically updated. Only the user knows which data files are associated with the table so it becomes his responsibility to maintain Index Data Sets when he changes a keyword table. The tool for accomplishing this function is the Index Specification utility and the method is to delete and add the index for all fields indexed with the updated table. The utility treats the deletion and addition of an index with identical parameters in the same run as a NO-OPERATION so indexes must be deleted in one run and added in a second run. An alternative is to rename the table when it is updated. Then the new table name is a new parameter when the index is added; deletion and addition can be accomplished in the same run. This method has the advantage of short circuiting indexes for the revised table which the user neglects to update; the old table name in these indexes will not be found. If the latter method is used, remember to rename all pages of the table. Table names are padded with EBCDIC zeroes to seven bytes and the eighth byte is used to number the pages. Page sequence is a blank, A through Z, then EBCDIC zero through nine. Each page is stored as a member in the user's library.

Note that only changes that affect an Index Data Set require user action. If words which do not appear in any form in a data file are deleted or added to a table, no action is necessary. Therefore, tables should be created or updated before data files are created or updated.

3.2.5 File Segmentation.

File segmentation permits a large SAM file to be subdivided into smaller subfiles, or segments. Each segment is treated much as an independent file, but all segments may be collectively processed as one main file, except in FM processing.

A segmented file must be subdivided by the contents of the high-order positions of the record control group. Each segment is defined by a range of values in the segment control field. File maintenance may be performed on only a single segment at a time, but retrieval and output operations may be performed on a single segment or on all segments through the use of the JCL data set concatenation capabilities.

File segmentation is a capability ideally suited to the maintenance of chronological files or other data files where data records are continually appended to the end of the data set. Segmentation minimizes the size of the data set to be processed in accessing any specific range of records.

3.3 File Field Specification

The user must consider a number of factors when defining file fields at file structuring time.

3.3.1 Control Field Considerations

The specification of file control fields is a task that is always performed but is frequently accomplished without proper analysis. Control fields to be specified are the record control group and subset control fields, if any, for each defined periodic set.

Other than limitations on length, the record control group must satisfy only one other requirement: to uniquely define each record in the file. When possible however, the record control group should be designed to aid in efficient operation of the file. The following steps should be considered in selecting the record control group:

a. Minimize the length of the control group. It must be included in every transaction and index (both ISAM indexes and secondary indexes). Extra characters can be costly.

field being printed. Only four one-byte fields can be LISTed on a single line of the 2260 scope if the user has defined 15-byte column labels. If no label is defined in the FFT, the field names should be as descriptive as possible, since they will be used as column labels by the QUIP LIST statement.

e. Direct Subset updating can be a very efficient method of updating a NIPS file. If direct subset updating is to be used in FM, however, a unique subset control field must be defined in the FFT.

3.4 Size Considerations

In designing and organizing a data file, the file designer should consider the importance of size in all aspects of the file. There are four major areas where size is important: total size of the file, data record size, set size, and size of the individual fields and groups.

3.4.1 File Size

File size becomes a problem whenever the file, because of its size, does not satisfy the response requirements imposed on it. Usually the failure is due to the number of file data records. This leads to extended run time for updating the file and producing the standard products.

Additional file size problems can arise as a result of the record size. Approaches to this problem are discussed in subsection 3.4.2. Also, extended run times can result from complex or inefficient employment of NIPS components (refer to Section 4, Language Characteristics) or, if the file is ISAM, from cylinder overflow from the prime area.

To reduce the number of records in a file, the first step is the elimination of all totally unnecessary records. Proper data record purge procedures should be defined and implemented as an integral part of the file. A logical purge may be used where records are automatically eliminated as a result of some data field value; e.g., a date, specific manual designation by a user, or some other external source, but procedures must be established.

Data records may be eliminated from a master file and retained on a second file as a history capability. The historical file may duplicate the format of the main file or it may utilize a different File Format Table.

Use of the purge and historical capabilities concurrently is also reasonable. Records would likely be purged from the main file as they are added to the history file. A possible history file procedure is shown in Figure 8. Two RASP queries are run against the master file defining purge and history specifications, respectively. The NIPS utility program UTQRTQDF is run against each RASP answer set to produce a purge file and a history file.

Initial file design can also forestall file size problems by concise definition of several smaller files rather than one large file. Utilization of NIPS capabilities such as IFO and merged file output could be used to bring data from the various files back together.

The file segmentation capability can be utilized to present smaller files to the file maintenance and retrieval/ output functions. Although sementation does not alter the size of a file, it permits the file to be processed as a series of subfiles, called segments. Refer to subsection 3.2.4.

File compression and compaction can also be used to reduce the physical size of the file. Compression and compaction provide a means of reducing intermediate storage requirements for data without altering the integrity of the data. This data reduction scheme is particularly suited to data files that contain strings of identical characters or a large amount of alphabetic data. The record control bytes of a record are not compressed or compacted.

# A string of four or more identical characters is compressed by translating it to two bytes. The first byte is a control byte which indicates that compression has been applied and gives a count of the number of identical consecutive bytes that were in the original string. The second byte is identical to those in the original string.

A string of alphabetic characters is compacted by translating it to a control byte followed by a string of coded characters. The control byte indicates that compaction has been applied and gives a count of the coded characters. Each

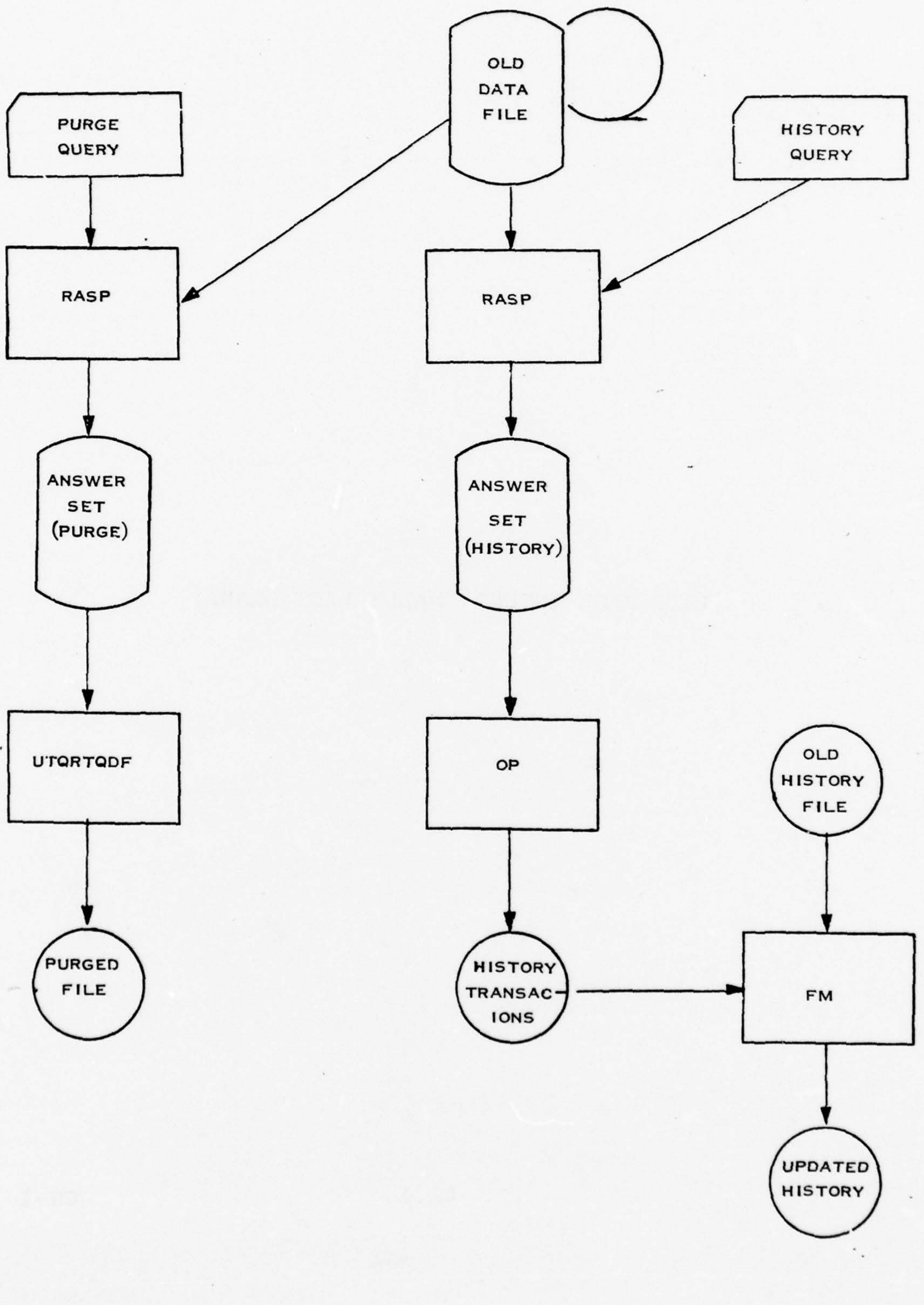


FIGURE 8. HISTORY FILE PROCEDURE CH-1

coded character represents a combination of two adjacent alphabetic characters.

Compression or compaction can be applied to a data file by specifying COMPRESS or COMPACT respectively as a value for the PARM parameter on the EXEC statement for the SAM to ISAM and ISAM to SAM utilities. The combination of both compression and compaction can be applied to a data file by specifying both keywords as values for the PARM parameter. When both are specified, compression is applied to a record first and those characters that cannot be compressed are processed for compaction.

The compression and/or compaction process can be reversed by specifying EXPAND as a value for PARM parameter for either utility.

The potential effectiveness of compression/compaction can be determined by executing the program NIPSDUMP with the PARM=CC option. This program provides statistics on file size, number and total size of each NIPS record type, and file size after compression compaction. No changes are made to the data storage mode when using PARM=CC option of the NIPSDUMP program. The following example shows the coding required to execute NIPSDUMP.

```
     // EXEC  PGM=NIPSDUMP,PARM=CC
     // STEPLIB DD DSN=FFS.JOBLIB,DISP=SHR
     // FILE    DD DSN=TESTER,UNIT=2314,DISP=SHR
     // SYSPRINT DD SYSOUT=A
     //SYSIN DD *

     /*
```

The following example shows the type of output produced by NIPSDUMP when the PARM=CC option is used.

```
     000348019 R-RECORDS IN FILE    SIZE=0028141560 BYTES
     000348019 RECORDS COMPRESSED   SIZE=0017834166 BYTES
     000347976 RECORDS COMPACTED    SIZE=0022259687 BYTES
     000348019 RECORDS BOTH         SIZE=0015835009 BYTES

          FOR COMPRESSION
     FILE SIZE REDUCED 36,.52 PERCENT
     R-RECORD (DATA PART ONLY) REDUCTION=44.96 PERCENT

          FOR COMPACTION
     FILE SIZE REDUCED 20.84 PERCENT
     R-RECORD (DATA PART ONLY) REDUCTION=25.66 PERCENT
```

FOR BOTH
FILE SIZE REDUCED 43.60 PERCENT
R-RECORD (DATA PART ONLY) REDUCTION=53.69 PERCENT

When processing compressed files, the record must be expanded when being read and compressed when being written. Very little processing is required to expand a record; file compresssion is a much more sophisticated routine. Therefore, reading a compressed file causes a very small increase in CPU time, writing a compressed record causes a larger increase in CPU utilization.

Since FM must both read and write updated records, the increase in CPU utilization will be greater for FM than for RASP, OP (source direct), and QUIP (source direct). During a SAM file update, every record must be read and rewritten. For an ISAM update, only the updated records will be processed. Therefore, file compression will have a relatively small impact on CPU utilization for ISAM file updates.

The number of factors affecting total processing efficiency of compressed files, requires that each application be reviewed independently to determine compression effectiveness for that application.

3.4.2 Record Size

Theoretically, a NIPS data record, because of its organization as a series of logical records, may be of any size. Certain NIPS components, however, must limit the size of the data record that can be retained for processing at one time. This limit, called the "process block size," is variable, depending on the component and the option of the user (see Introduction to File Concepts, "Data Record Organization Summary", CSM UM 15-74). Process block size limitations are discussed in appendix C of this document.

Because of the process block limitations and because large records greatly increase file storage requirements, record size often becomes a problem.

Data record size may be reduced by data condensation techniques such as subroutine or table conversion. However, the user must weigh the advantages and costs of data conversion. Some of the factors to be considered are:

a. Effort required to develop, maintain, and use conversion tables or subroutines. Whether or not conversion techniques are standardized and meaningful to all users.

b. Full Size--Small files probably do not merit the effort required to implement conversion techniques.

c. Reduction attained--A reduction from 20 bytes to two bytes is more significant than from three bytes to one byte.

d. Frequency of occurrence--Converting a record control group field is more effective than converting a fixed set field, because the record control group is included in each logical record.

e. Conversion time--If the transaction data exists in coded form, moving it into the file in that form eliminates FM conversion. If the transaction is not coded, FM conversion would be required to store the data in coded form.

f. Accuracy for retrievals--Spelling and keypunch errors may be reduced if a code can be specified in lieu of a 20- or 30-character name, which may be input in various formats.

3.4.3 Set Size

The set is the highest NIPS structure with a size limitation. The maximum size for the fixed set or, for a subset of any periodic set, is 1000 bytes or 100 defined fields and groups. The format of the NIPS record is defined in Appendix A of NIPS User's Manual, Volume I - Introduction to File Concepts.

Fixed data is the simplest data to process in NIPS. Thus, it is advantageous to store all nonrepetitive data in the fixed set. If, however, the set size limitations are exceeded, fixed data could be stored in a periodic set. If this is done, it is best to define as periodic those data items most likely to be blank. When they are blank, that storage space will not be required. This technique could be employed to save space even if the original set size were not exceeded.

has shown that processing queries serially against a single file is significantly faster than attempting parallel queries.

Multiple data sets on the same pack also cause device contention if both data sets are queried at the same time. Ideally, each NIPS file in the TP environment should be on a separate disk pack. This, however, is impractical, since the number of available disk drives would quickly limit the number of files that could be mounted at any time.

It is possible for a NIPS file library to contend with its associated data file if resident on the same pack. This should not be a significant problem unless lack of free core requires frequent rolling of tables and subroutines. If the tables and subroutines can remain resident throughout the query, then library access and resulting contention should be minimal.

An Index Data Set does not compete with its associated file when mounted on the same pack. Accesses to the Index Data Set and to the master file are performed serially by QUIP. Processing of the Index Data Set is completed prior to reading the data records from the master file.

Online file access can be improved by allocating the file index and prime data area to separate devices. The prime data area can also be allocated to several separate packs. These techniques reduce the contention caused when multiple terminals access the same file concurrently.

5.1.3 Terminal Versus Batch Applications

In many applications, the user has the choice of running in the batch or TP mode. Wherever possible, terminal queries should be limited to relatively short, precise portions of the data base. Sorting of output should be limited.

Stored queries that are run on a regular basis and have output dumped to the printer should be initiated as batch jobs by the remote job entry (RJE) feature of EDIT. Relieving TP of a heavy load of production type TP queries improves the TP responsiveness to ad-hoc queries.

5.1.4 Restricting File Search

Two NIPS capabilities can be used to improve TP response. These are LIMIT and secondary indexing.

LIMIT restricts search of the file to records that fall within the range of the LIMIT statement. If the upper range of the LIMIT includes the last record in the file, ISAM reads all pad records (if present) until it reaches the end-of-file. When using the LIMIT statement in this manner, it is important to use the NOPAD option when creating the file.

Secondary indexing restricts search of the file to those records that can possibly satisfy the query, based on evaluation of clauses containing fields specified as index fields. The secondary indexing capability is discussed in section 3 of this document.

5.2 Terminal Operating Techniques

This subsection discusses a number of techniques which can be employed by the terminal user to improve overall TP operation or to circumvent potential user errors.

5.2.1 General TP Procedures

The LOGON, LOGOFF requirements may vary by installation. As a minimum, the user should include his name when logging on or off. Also, any problems encountered during the terminal session should be recorded using the REMARKS request. For example:

```
REMARKS UNIT BUSY 1300-1320-JONES.
```

Information from the LOGON, LOGOFF, and REMARKS requests is recorded on the TP Log (DDNAME=STATRECS). This information assists systems programmers in isolating any problems that might occur.

As soon as the user is logged on, he should send a message to the operator requesting mounting of the desired pack(s). For example:

```
PLEASE MOUNT PACK 123456-JONES. ¬M
```

Similarly, he should tell the operator when he has finished.

```
I'M FINISHED WITH PACK 123456-JONES. ¬M
```

The user should verify that his data base is on the desired pack prior to starting the terminal session. The following sequence has frequently caused deletion of a production data base from its pack:

a. UTBLDSAM to create tape file

b. Delete file from disk pack

c. Perform production update on tape file

d. Restore new file to disk using UTBLDISM.

d. Add Basic JCL from Source Library

```
          /GET MEM=JCL1 LIB=TEST360L
          /LIST ¬E
         EOM RECEIVED
         EDIT PROCESS COMPLETED. START CONVERSATION.

         N

         RECORDS      0000 THRU      END  OF EDIT WORK FILE
         //TP999901 JOB (555,1234,12,U,0000),JONES6633,
         //   CLASS=C,REGION=100K
         /*VOLUMES ND1234
         /*SCHEDULE TAPE9=0 TAPE7=0 DISK=1
         //ABC  EXEC XQUIPSD,ISAM=TEST360,VISAM='SER=ND1234,
         //   LIB=TEST360,VLIB='SER=ND1234',CL=U,CL1=U,CL2=U
         //SYSIN DD *
         FILE TEST360. CLASS UNCLASSIFIED
         LIMIT SERV = NAVY.
         LIST UIC PERS MEQPT
```

e. Tailor JCL for this run

```
          /C 1 *9999*0205*
          /RESEQ
          /L¬E
         EOM RECEIVED
         EDIT PROCESS COMPLETED. START CONVERSATION.

         N

         RECORDS      0000 THRU      END  OF EDIT WORK FILE
         //TP020501 JOB (555,1234,12,U,0000),JONES6633,
         //   CLASS=C,REGION=100K
         /*VOLUMES ND1234
         /*SCHEDULE TAPE9=0 TAPE7=0 DISK=1
         //ABC  EXEC XQUIPSD,ISAM=TEST360,VISAM='SER=ND1234',
         //   LIB-TEST360,VLIB='SER=ND1234',CL=U,CL1=U,CL2=U
         //SYSIN DD *
         FILE TEST360. CLASS UNCLASSIFIED
         LIMIT SERV = NAVY.
         LIST UIC PERS MEQPT
```

f. Submit Job for Batch Execution

```
          /SUBMIT ¬E
```

5.2.5 Access to EMQ and OMQ

The /LIST operator in EDIT places data on the OMQ, if no EDIT errors are encountered. EDIT advisory messages are placed on the Edit Message Queue (EMQ). After successful execution of LIST, the user receives the START CONVERSATION message. At this point, he can communicate with either the OMQ or the EMQ. He selects the OMQ by using the paging command N. He selects the EMQ by using the paging command E. All subsequent paging operations will be applied to the queue selected. To review the other queue, the user must scratch or hold the queue he is working with and execute the PAGE program (¬P). He may then access the alternate queue.

#5.2.6 Record ID Retrieval

SODA should be considered as a retrieval and output tool for terminal applications. If the record ID of the requested record is known, and a specific output format is required, SODA can be used to display required data from a single record. SODA provides extremely fast response because of two factors:

a. The record ID of the desired record is known and no file search is needed.

b. SODA executes a precompiled logic statement and requires no translation phase. A QUIP query must be translated before it is executed.

The record ID and logic statement ID entered at the terminal is processed by a user-written logic statement which formats the output display.

#5.2.7 VIEW Program

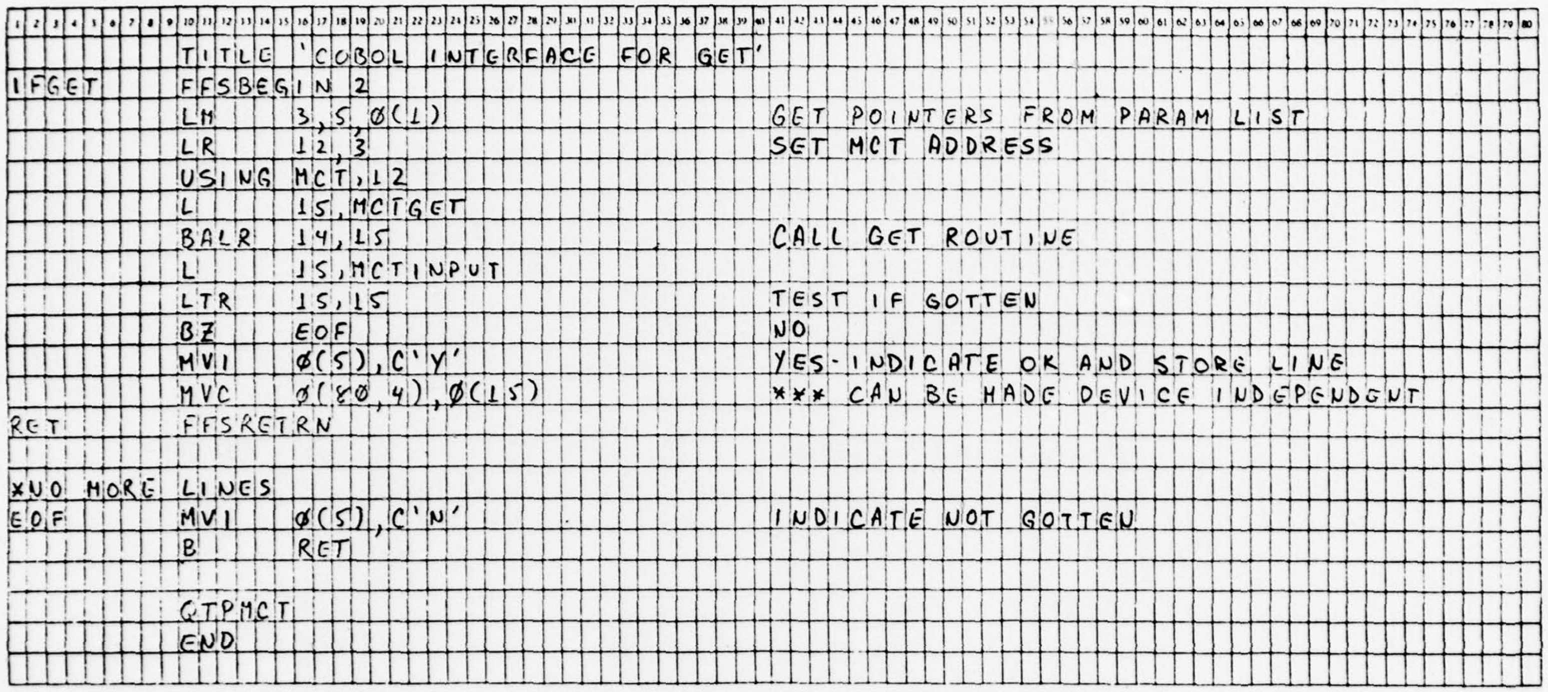
Many standard batch and online output requests can be anticipated following the file update. The production of predefined reports can be accomplished as part of the regular file update cycle, and the outputs can be saved on distribution data sets.

Once the output reports are stored on a distribution data set which is accessible by the terminal, the outputs can be selected and viewed at the terminal as required.

The VIEW program formats a list of output report titles (often called a menu) and allows the user to page through the list and select a report to be viewed. Thus users not familiar with NIPS TP and the QUIP query language can choose an output from the menu and page through it with a few simple commands. This capability makes the data accessible to the decision making personnel. In addition, response is almost instantaneous, since no data base search is required.

Use of distribution data sets and VIEW should be considered as an alternative to producing batch reports which are required as reference material by multiple users. The convenience and flexiblity of terminal access using VIEW, as well as the savings in printing and distribution costs can, significantly improve operating efficiency.

The use of the VIEW capability is discussed further in section 2.7 of CSM UM 15-74, Volume VI - NIPS Terminal Processing (TP).



```
         TITLE 'COBOL INTERFACE FOR GET'
IFGET    FFSBEGIN 2
         LM    3,5,Ø(1)                 GET POINTERS FROM PARAM LIST
         LR    12,3                     SET MCT ADDRESS
         USING MCT,12
         L     15,MCTGET
         BALR  14,15                    CALL GET ROUTINE
         L     15,MCTINPUT
         LTR   15,15                    TEST IF GOTTEN
         BZ    EOF                      NO
         MVI   Ø(5),C'Y'                YES-INDICATE OK AND STORE LINE
         MVC   Ø(80,4),Ø(15)            *** CAN BE MADE DEVICE INDEPENDENT
RET      FFSRETRN

*NO MORE LINES
EOF      MVI   Ø(5),C'N'                INDICATE NOT GOTTEN
         B     RET

         GTPMCT
         END
```

Figure 30. COBOL Interface With GET, PUT and EXIT (1 of 3)

```
         TITLE 'COBOL INTERFACE FOR PUT'
IFPUT    FFSBEGIN 2
         LM    3,6,Ø(1)                 GET POINTERS
         LR    12,3                     SET MCT ADDRESS
         USING MCT,12
         CLI   Ø(5),C'Q'                IS IT FOR OUTPUT QUEUE
         BNE   IMMED                    NO
         ST    4,MCTOUTPT               YES-PASS TO PUT ROUTINE
         MVC   MCTCLASS,Ø(6)                  AND PROVIDE FOR CLASSIFICATION
         L     15,MCTPUT
         BALR  14,15                    PASS TO PUT ROUTINE
RET      FFSRETRN
* IMMEDIATE OUTPUT
IMMED    ST    4,MCTINOUT               PASS TO TPIO ROUTINE
         CLI   Ø(6),C'Y'                IS RESPONSE REQUESTED
         BNE   *+8                      NO
         MVI   MCTINOUT,X'FF'           YES-TELL SUPERVISOR TO DO IT
         L     15,MCTTPIO
         BALR  14,15
         L     4,MCTINOUT

         QTPMCT
         END
```

Figure 30. COBOL Interface With GET, PUT and EXIT (2 of 3)

It is possible to write a COBOL (or FORTRAN or PL/1) interface routine to perform interterminal output. Needed parameters are the terminal name, and an 01 record layout that begins with 02 USTADDR PICTURE S99999 COMPUTATIONAL. This will leave room to store the UST address. The output data must follow as the second field.

INDEX

INDEX

CH-1

INDEX

CH-1

INDEX

INDEX

INDEX

CH-1

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX

INDEX