To: MTB Distribution

From: C. Erickson and J. Falksen

Date: 15 October 1976

Subject: Indexing Manuals on the Multics System

This MTB describes a new method for generating indexes for manuals on the Multics system.

Old Methods

The indexes to the FORTRAN and BASIC manuals were generated by a set of programs that used a driver file to produce index "hits". A "hit" occurred whenever the program encountered a driver file term in the text. However, the false hits far outnumbered the genuine ones (consider "do" and "where") when these programs were tried on the MPM.

Therefore a second indexing scheme was used on the MPM that identified where a hit should occur by actually modifying the runoff source. This second indexing technique provided the basis for the new method described in this MTB. The new method includes topic, module, "see", and concept hits and can also provide a side-by-side runout (text and indexed items) for final review.

Objectives

One of the main reasons this indexing method was devised was to ensure uniform appearance of indexes within the Multics set of manuals. Also, if the indexes are all produced in the same manner, a comprehensive index can be produced without too much effort.

Since the GCOS manuals are now produced on the Multics system, it was necessary that the new indexing method accept as input the indexing marks already embedded in the runoff source of the GCOS manuals.

Another main objective was to mimimize the human effort required in producing an index. Any indexing method requires a great deal of human time deciding how an index should be organized, what should be in it, how items should be included, etc. The best the computer can do, until it understands the English language, is make sure the human time is expended only once. By modifying the runoff source segment to identify where and how an index entry should occur, the index can be generated over and over again and the entries will continually be produced as they were the first time (except for the page numbers, which runoff will keep track of). Thus, for any updates, human indexing time is reduced to deciding on index entries for the new material.

The new indexing method offers a side-by-side feature, which will probably be used on final review copies, to check the index hits against the text. The left portion of the dprint is the text of the manual; the right portion consists of index entries, placed as close as possible to the corresponding text.

Finally, the last main objective (not yet a reality) is to have all the programs/macros used in the indexing process installed in the tools library. This status will ensure a certain level of support and make the information more readily available to the general user community.

Index Format

The new method produces an index that looks like this:

```
command language
  execution
    abbrev  3-3
    answer  3-12
    enter_abs_request  3-126
    exec_com  3-130
    memo  3-209
    walk_subtree  3-335
  expansion
    abbrev  3-3
    do  3-100
    get_com_line  3-166
    set_com_line  3-313
  see active functions
  see directory
  see search rules
```

Notice that different levels of headings are allowed. Up to four levels of headings can be accommodated, and any level can be followed by a page number.

Implementation

The new method of indexing proposed in this MTB ensures that index entries, once defined, are constant. This method also minimizes the amount of extra typing necessary to generate a hit; i.e., if the key string actually appears in text, it need not be retyped. For example, consider the following index entry:

```
command language
  execution
    abbrev  3-3
```

This entry could be produced by the following lines in the runoff segment:

```
The
.if hit "Kabbrev|command language~execution"
command
```
OR
```
The ^Kabbrev|command language~execution@ command
```

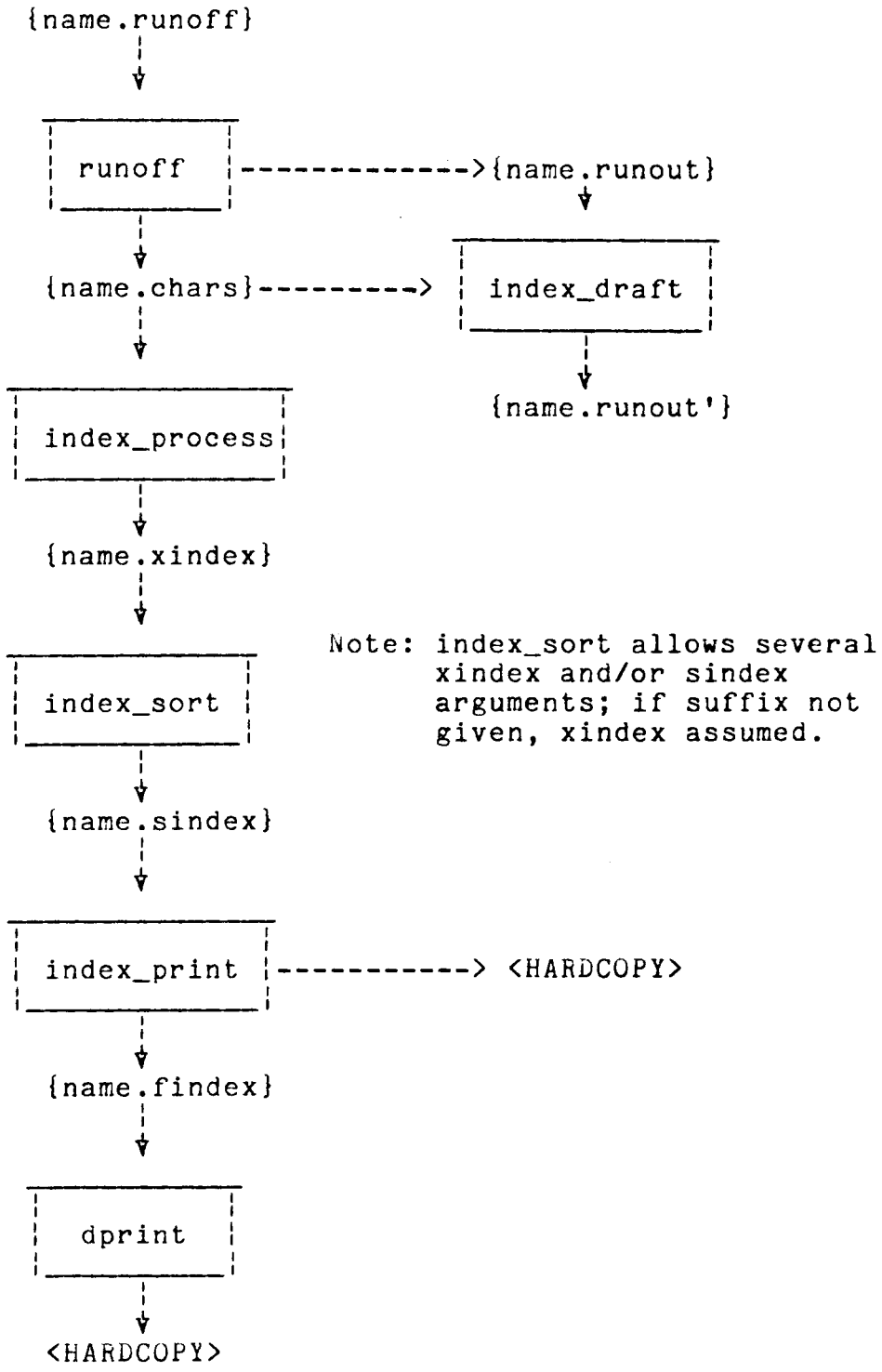
These lines are examples of the two methods of indexing manuals on Multics: the hit include file method and the marking method.

The first step in the indexing process is to modify the runoff source using one of these methods. Most Multics manuals will use the include file method; the marking method is similar to the indexing procedures originally used on Series 60 Level 66 (GCOS) manuals and is provided mainly for use by GCOS manual production personnel. (The marked lines in a runoff segment must be transformed into hit lines, currently done using a ted_com, before any further indexing procedures can be done.)

Overview of Indexing Process

When the segment containing hit lines is run off, an entry for each hit line is put in the chars segment and, if specified, text is passed to the runout segment (or printed if the -sm control argument is not used). The index_process (ted_com) program then takes the hit lines in the chars segment, permutes them to form keys, strips off extraneous page information, and puts the result in the extracted index (a segment with the suffix, xindex). Another program, index_sort (ted_com), sorts the lines in the extracted index to produce the sorted index (suffix, sindex). Then, the index_print program creates a formatted index (suffix, findex) from the sindex segment. This is the finished index. To produce the side-by-side version of the document for final review, the index_draft program takes the chars segment and the runout segment and produces a two-columned runout segment (runout'), which is then dprinted.

The diagram below illustrates the indexing process.



Detailed Description

The remainder of this MTB explains how the modification of the runoff segment is done and gives several examples, showing sample runoff hit lines and the result in the runout, chars, and xindex segments. The indexing procedures briefly described here must be fully documented for future use by programmers, writers, and terminal operators. Also, as mentioned previously, the programs involved must be documented and submitted to the tools library. If necessary, a future MTB will deal exclusively with a detailed walk-through implementation of the indexing procedures.

If the marking method (as opposed to the hit include file) is used to modify the runoff segment, the begin and end marks (^ and @ in the example above) should be characters that do not appear anywhere else in the manual.

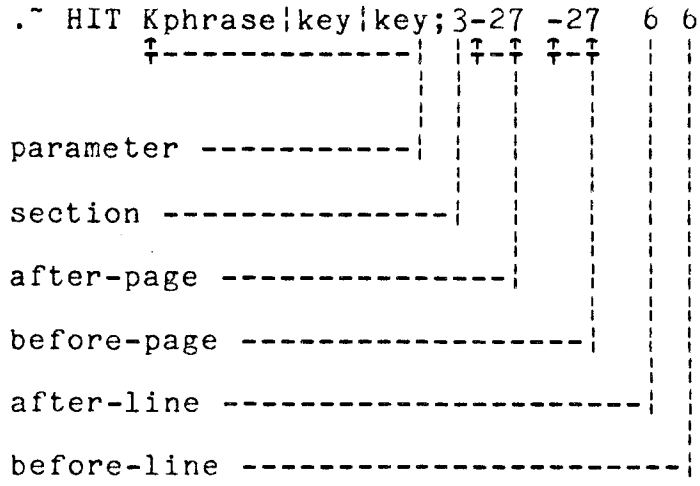
If the hit include file method is used, the quote character (") cannot be used as part of the hit.

The following characters cannot be used as part of the hit; they are reserved for special use in the parameter to the hit macro:

~	used to separate levels
¦	used to separate phrase from key
;	used to separate page info from rest of string

The forms of hits that cause text to appear in the runout segment cannot be used within nofill (.nf), center (.ce), or equation (.eq) modes. An error message will occur if this is attempted.

The index hits are collected in the chars segment (to avoid various initialization problems and difficulty with the command processor). A hit line in the chars segment will look like this:



Line numbers and page numbers are needed for generation of a side-by-side final review version of a document.

Various kinds of hits can be specified so that the writer can control:

1. hit format (uppercase, lowercase, initial caps, or as-is)

2. hit formation (either writer explicitly gives keys or programs generate keys by permuting text portion of hit parameter)

3. text generation (whether or not text appears in the runout segment)

4. "see" items

5. "no-index" items

The control character at the beginning of the hit parameter (e.g., K in the hit line above) determines the actions taken by the indexing programs.

1. Text is generated and permuted to form keys.
   "Utext" uppercase keys
   "Ltext" lowercase keys
   "Itext" initial caps keys
   "Atext" as-is keys
   "+text" uppercase keys and page_no* (used to indicate new index entries, usually for addenda)

2. No text; string is permuted to form keys.
   "U|string" uppercase keys
   "L|string" lowercase keys
   "I|string" initial caps keys
   "A|string" as-is keys
   "+|string" uppercase keys and page_no*

3. Text is generated and keys are specified.
   "Ktext|key|key...|key"

4. No text; keys are specified.
   "K|key|key...|key"

5. No text; keys are specified; no page reference in index.
"S|key|key...|key"

6. Text is generated and no entry appears in index.
"Ntext"
(This type of hit is used when a word or phrase that appears in text should not be indexed here even though it is indexed elsewhere in document.)

Examples

```
line in .runoff:    .if hit "UABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT UABC Def ghi;3-27 -27 6 6

result in .xindex:  ABC~ABC Def ghi;3-27
                    DEF~ABC Def ghi;3-27
                    GHI~ABC Def ghi;3-27


line in .runoff:    .if hit "+ABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT +ABC Def ghi;3-27 -27 6 6

result in .xindex:  ABC~ABC Def ghi;3-27*
                    DEF~ABC Def ghi;3-27*
                    GHI~ABC Def ghi;3-27*


line in .runoff:    .if hit "NABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   NONE

result in .xindex:  NONE


line in .runoff:    .if hit "LABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT LABC Def ghi;3-27 -27 6 6

result in .xindex:  abc~ABC Def ghi;3-27
                    def~ABC Def ghi;3-27
                    ghi~ABC Def ghi;3-27
```

```
line in .runoff:    .if hit "IABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT IABC Def ghi;3-27 -27 6 6

result in .xindex:  Abc~ABC Def ghi;3-27
                    Def~ABC Def ghi;3-27
                    Ghi~ABC Def ghi;3-27


line in .runoff:    .if hit "AABC Def ghi"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT AABC Def ghi;3-27 -27 6 6

result in .xindex:  ABC~ABC Def ghi;3-27
                    Def~ABC Def ghi;3-27
                    ghi~ABC Def ghi;3-27


line in .runoff:    .if hit "KABC Def ghi|first"

result in .runout:  ABC Def ghi

result in .chars:   .~ HIT KABC Def ghi|first;3-27 -27 6 6

result in .xindex:  first~ABC Def ghi;3-27


line in .runoff:    .if hit "Kruns|second|third~inning"

result in .runout:  runs

result in .chars:   .~ HIT Kruns|second|third~inning;3-27 -27 6 6

result in .xindex:  second~runs;3-27
                    third~inning~runs;3-27


line in .runoff:    .if hit "K|third"

result in .runout:  NONE

result in .chars:   .~ HIT K|third;3-27 -27 6 6

result in .xindex:  third;3-27
```

line in .runoff: .if hit "K¦ball~1¦strike~2"

result in .runout: NONE

result in .chars: .~ HIT K¦ball~1¦strike~2;3-27 -27 6 6

result in .xindex: ball~1;3-27
strike~2;3-27

line in .runoff: .if hit "S¦ACL~see access"

result in .runout: NONE

result in .chars: .~ HIT S¦ACL~see access;3-27 -27 6 6

result in .xindex: ACL~see access

line in .runoff: .if hit "S¦invoke~see INV¦PRn~see register, procedure" (this would actually be only one line)

result in .runout: NONE

result in .chars: .~ HIT S¦invoke~see INV¦PRn~see register, procedure;3-27 -27 6 6 (again, really only one line)

result in .xindex: invoke~see INV
PRn~see register, procedure