Automated Index Generation for LaTeX

Richard L. Aurbach
Monsanto Company
St. Louis, Missouri

Abstract

LaTeX includes partial support for the generation of an Index in a document. It contains commands which enable index terms and the pages on which they appear to be captured in an auxiliary file. However, special processing (external to LaTeX itself) is required to translate this information into a pleasingly-formatted index.

The IdxTeX program provides this additional processing, and generates a file of LaTeX source which may be included in a document to produce the desired index.

This paper describes how IdxTeX provides a full range of services for index generation and discusses issues related to the development of programs which provide auxiliary processing for LaTeX documents.

The IdxTeX Project

The LaTeX text formatting program is an extremely versatile tool for the generation of high-quality documents. However, its handling of an Index is incomplete. LaTeX provides the `\index` command which (in conjunction with the `\makeindex` command) generates an auxiliary file containing index terms and references to the pages on which the terms appear. However, it is left to the writer to develop this information into an appropriately formatted Index.

As part of a project to develop new document styles, I became interested in the automation of this process of Index generation and developed the IdxTeX program[1] to complement the capabilities of LaTeX. The IdxTeX project had several goals:

- to provide a fully-automated mechanism for Index generation which produces an Index with the same level of quality as the LaTeX document in which it appears.
- to help make indexing sufficiently easy to encourage authors to build effective and helpful indices into their documents.
- to provide a full set of indexing features, so that even complex indices (such as the Index of *The TeXbook*[2]) could be generated.

[1]The IdxTeX program has been submitted to both the TeX Users Group and to the DECUS Library for distribution to interested parties. The distribution includes a Users Guide, which describes how to use the program, an executable VAX/VMS image, and complete sources in the C language. Since the program uses VAX/VMS services, it will only run in that environment. However, I believe that it could be ported to other environments with modest effort.

[2]Knuth, Donald E., *The TeXbook*, Addison-Wesley and the American Mathematical Society, 1984.

- to provide support for all of the indexing capabilities inherent in LaTeX, such as three-level indexing, without requiring any additional enhancements to LaTeX itself.
- to include support for the generation of a Master Index for a set of documents.

The Indexing Problem

Generating an index in the LaTeX context provides a number of interesting challenges.

Index Levels	LaTeX supports a three-level index (items, subitems, and subsubitems). However, the `\index` command accepts only one argument. It is necessary to adopt a convention within the text of its argument to specify the level of the term being indexed. In IdxTeX, the `>` symbol is used to separate the item from the subitem and the subitem from the subsubitem.
Spelling	Obviously, the index must appear in alphabetical order. However, it must be possible to include LaTeX commands within an index term, to optimize the visual appearance of the Index. That is, an author should be able to specify "`\index{{\em Special\/} Commands}`", for example, and have the index item appear as expected. This means that IdxTeX must understand LaTeX syntax, so that the term can be properly placed in alphabetic order.
Page Ranges	If an item is indexed on a series of consecutive pages, the index entry should display the range of pages, rather than a list of consecutive numbers. That is, an item which is indexed on pages 11, 12, and 13, for example, should appear in the index with a page reference of 11–13.
Cross References	It is not uncommon to see an item in an index which refers to one or more other items in the index. To support this, syntactic conventions in the `\index` command and special processing are necessary.
Master Index	To generate a Master Index, the program must be able to process more than one auxiliary file, and keep track of which volume of the volume set is associated with each item. The output of the program must include labels which identify the volume associated with each index item.

The following sections provide insights into how each of these issues was resolved in IdxTeX.

Indexing Conventions

The LaTeX `\index` command takes a single argument. In an automatic index generation environment, that argument represents the only mechanism by which the author can communicate information to IdxTEX about how the term should be handled. To allow for the multitude of index features supported, it was necessary to impose a set of conventions on the use of this command.

Two principles were important to the design of these conventions:

1. The conventions should be (as much as possible) mnemonic, so that they are easy to remember.

2. The conventions should be easily recognizable as such. That is, the program must be able to distinguish unambiguously between characters which are used as part of a convention and characters which are part of the term being indexed.

Conventions were chosen which are not valid LATEX syntax — they would generate LATEX errors if they occurred naturally. Since IdxTEX is sensitive to LATEX syntax, this assures that there will be no cases in which IdxTEX confuses a part of its conventions for legitimate text entry.

The following conventions are used in IdxTEX.

Level Separators

The > character is used to separate items from subitems and subitems from subsubitems.[3] For example,

```
\index{Aaa>Bbb>Ccc}
```

specifies an index entry with an item of "Aaa", a subitem of "Bbb", and a subsubitem of "Ccc". Of course,

```
\index{Aaa>Bbb}
```

or

```
\index{Aaa}
```

are also acceptable.

Page Reference Highlights

The first character of an index entry may be used to specify special formatting for its page reference. The following table lists the capabilities which are available.

Format	**Meaning**	**Example**
`\index{^Foo}`	**boldface**	Foo, **11**
`\index{_Foo}`	underline	Foo, 11
`\index{~Foo}`	*italics*	Foo, *11*
`\index{#Foo}`	"and following"	Foo, 11ff

Cross References

Cross references are specified using the & character. For example,

```
\index{Aaa&Bbb}
```

will generate a cross reference of the form

"Aaa, *see* Bbb"

[3]Note that the RUNOFF text formatting system uses the same convention. Since we expected to convert a number of RUNOFF documents, this choice was obvious.

Cross reference processing allows for a combination of real page references and cross references, so that a combination of entries such as

```
\index{Aaa} \index{Aaa&Bbb}
```

will generate

"Aaa, 11, *see also* Bbb"

Master Index Master index processing uses a new type of auxiliary file to provide the information IdxTEX needs to understand which document indices to use when building the Master Index and what labels to use when displaying information from different volumes. This will be discussed in more detail below.

Data Structures

The Index of a large document or the Master Index of a large document set may be quite extensive. To avoid limitations on the number of items which IdxTEX could handle, all internal data structures are allocated from dynamic memory. Therefore, the size of an Index is limited only by the user's virtual page quota.

Since the three-level structure of the index implies a tree-like organization, the basic data structures selected for internal storage of index information in IdxTEX were linked lists. While linked lists are not optimally efficient in this application, their simplicity compensates for the minor loss of performance.[4]

The basic data structure for each index item, subitem, or subsubitem is called a NODE. Using the notation of the C language, a NODE can be defined as

```
typedef struct node
    {
    struct node           *link;
    struct dsc$descriptor item;
    struct dsc$descriptor spell;
    struct node           *subhead;
    struct pgnode         *pghead;
    struct pgnode         *cfhead;
    } NODE;
```

In this structure, *link* is the forward linkage pointer to the next node in the list; *item* is a VAX/VMS dynamic string descriptor[5] which describes the text string associated with the index item; and *spell* is another string descriptor for the *spell-string*. The *spell-string* is used when alphabetizing index

[4]Since IdxTEX is not run often, its cost is an inconsequential fraction of the total cost of generating a document.

[5]VAX/VMS dynamic strings were used (rather than the ASCIZ strings which are more natural in a C-language implementation) because the VMS services which work with them handle all details of dynamic memory allocation and deallocation.

entries and helps resolve the spelling problems discussed previously. It is discussed in more detail below.

The *subhead* is the listhead for a linked list of NODEs for any subitems associated with this index item. The recursive nature of this data structure made handling the three levels of indexing simple.

The *pghead* and *cfhead* variables are listheads for linked lists of PGNODE structures. Each PGNODE structure includes information about a single reference to the particular index entry. The list chained from *pghead* contains numeric page references, while the list chained from *cfhead* contains cross references.

Using C language notation, a PGNODE has the following structure

```
typedef struct pgnode
    {
    struct pgnode          *link;
    struct dsc$descriptor  *vol;
    struct dsc$descriptor  page_dsc;
    char                   highlight;
    } PGNODE;
```

Once again, *link* is the forward linkage pointer for the linked list. The *vol* variable is used in Master Index processing to point to the dynamic string descriptor for the label to be associated with the volume from which the reference came. The *page_dsc* describes the page reference string, while *highlight* is a flag used to indicate what type of page reference highlighting is associated with this page reference.

One virtue of this type of internal data organization is that each distinct item, subitem, or subsubitem uses only a single NODE structure. If the entry has a number of page references, then one PGNODE structure (which is fairly small) is used for each. If more than one index reference occurs on the same page, only a single PGNODE is allocated. This approach conserves dynamic memory.

NODEs are linked together in alphabetical order (by *spell-string*). PGNODEs for numeric page references are linked together in the order they appear in the auxiliary file produced by LaTeX, which automatically puts them into numerical order. PGNODEs for cross references are linked together alphabetically. This means that the internal representation of the index is built in sorted order, simplifying back-end processing.

Spell Strings and Alphabetization

As discussed previously, putting index entries into alphabetical order is a complex task, because the entry may contain LaTeX commands which are meant to enhance the visual appearance of the index, but which must not be included when the term is placed in alphabetical order. In IdxTeX, the concept of a *spell-string* was introduced to handle this problem.

The basic idea is that each NODE of the internal data structure contains descriptors for two copies of the index entry — the *item* and the *spell-string*.

- The *item* string contains the original form of the index entry, including all LaTeX commands. It is used to generate the formatted output and is not used when placing the entry in proper alphabetical order.

- The *spell-string* originally contains a copy of the index entry. However, during spelling processing, it is modified to remove everything which should not be included when the entry is placed in alphabetical order. It is not used for any other purpose.

Therefore, spelling processing consists of a number of steps which recognize various forms of LaTeX syntax and remove them from the *spell-string*. After this has been done, the *spell-string* is in a form suitable for alphabetizing the index entry, while the *item* string remains untouched.

In some special documents, it may be desirable to place TeX or LaTeX commands themselves in the index.[6] To accommodate this possibility, spelling processing skips any text contained within a `\verb` or `\verb*` construct. This means, for example, that

```
\index{\em Command}
```

will be treated as if it were spelled as "`Command`", but

```
\index{\verb+\em+ Command}
```

will be treated as if it were spelled as "`\em Command`".

The spelling processing performs the following operations (in order)

- Accents are processed. All of the special characters associated with the accents are removed. For example, in the *spell-string*, `se\~{n}or` is translated to `senor`.

- Emphasis commands are removed from the *spell-string*. Examples of emphasis commands are `\rm`, `\bf`, `\large`, etc.

- Grouping and mode commands are removed from the *spell-string*. That is, `{`, `}`, and `$` are removed. However, `\{`, `\}`, and `\$` are retained, since they do not represent grouping commands.

- Backslashes are removed from the *spell-string*. The logic which skips processing in `\verb` and `\verb*` constructs prevents the "`\`" in "`\verb`" from being removed.

- `\verb` and `\verb*` constructs are cleaned up. For example, "`\verb+foo+`" is translated to "`foo`".

- The *spell-string* is converted to upper case, all unnecessary whitespace is removed, and a few minor corrections are made to handle special cases.

  For example, the *spell-strings* of index items which begin with non-alphanumeric characters are adjusted so that all such terms will appear in the index before any items which begin with any alphanumeric character.

[6]Obvious examples of this are the indexes of documents about text processing.

Also, special logic is used to assure that any index term which begins with a `\verb` or a `\verb*` is placed in the proper place in the index. This includes adjustments to the *spell-string* which prevent references for items such as "`input`" and "`\verb+\input+`" from being confused.

This approach has proven to be effective in developing an index which uses LaTeX commands liberally, but retains proper alphabetical order. There are, however, aspects of spelling processing which can be debated.

- In *The TeXbook*, native TeX commands are displayed with a leading asterisk, but are alphabetized as if the asterisk were not present. IdxTeX does not currently handle this case.
- IdxTeX is case blind. That is, `\index{Large}` and `\index{large}` are considered two instances of the same item.[7] The case displayed in the index matches that of the first item seen. This is usually desirable — it prevents some typographical errors from generating unwanted index entries. However, there may be some cases in which case sensitivity would be preferred.

Page Ranges

Another issue which appears simple, but has a number of interesting complications is the handling of page ranges. Indeed, the simplest case (converting references on pages 11, 12, and 13, for example, to a reference to "11–13") does not present any significant difficulties. However, the general case is not that simple.

- Since we support page reference highlighting, it is necessary that the system recognize that 11, 12, and 13 constitute a page range, but that **11**, *12*, and <u>13</u> must be handled differently.
- A reference such as "20ff" represents a different type of page range. If a term is also indexed on page 19, then the index entry should read "19ff" rather than "19, 20ff".
- Some document styles use chapter oriented (or other complex) page numbers. The algorithm which determines whether pages are adjacent must be able to handle page numbers such as "5–2" or "Glossary–4".
- In a Master Index context, the algorithm must also be able to determine that a reference to page 11 from Volume I is not adjacent to a reference to page 12 in Volume II.

It turns out that solving these complications is unreasonably difficult during the initial building of the internal data structures. Therefore, a special processing step is used to handle page ranges.

For each linked list of page references, an array of special data structures[8] is dynamically allocated and the information from the linked list is moved to the array. Each page reference text string is parsed into a volume string, a chapter string (if any), a page number, and a highlight flag. Two pages are adjacent if they have the same volume, chapter, and highlight, and consecutive page numbers.

[7] In fact, **any** items which have the same *spell-string*, according to the syntax rules above, will be considered instances of the same item. The displayed text will be that of the first index reference seen.

[8] My thanks to my colleague, Donald R. Gummow, for suggestions concerning this internal array.

Page references which are parts of ranges are flagged as the beginning, middle, or end of the range. Since the "*and following*" notation is handled internally as a highlight, it is relatively easy to handle special cases involving this type of page reference within a page range.

Once this analysis is complete and one or more page ranges is discovered in the list of page references, the initial list is deleted and a new page reference list is built based on the information in the array. Since this approach concentrates all of the page range logic in one place, the routines which format the output require no special logic. Also, given the amount of information stored in the array, it is trivial to provide special touches, such as formatting a range of simple page numbers as "`11--13`", while handling a range of complex page numbers as "`2--6 to 2--10`".

Other Features

A number of other features of the program deserve some mention.

Cross References Handling cross references turned out to be surprisingly easy, once I realized that they should be segregated from page number references in their own linked list. This allowed multiple cross references to be listed in alphabetical order, and eliminated problems associated with mixtures of page number references and cross references for the same term.

At present, IdxTEX does not check to verify that an index entry actually exists for each cross reference, but this desirable feature could be added without great difficulty.

Master Index Some special processing is required to generate a Master Index.

- There must be a mechanism to inform IdxTEX of which auxiliary files to process to build the Master Index (and in which order to process them). This problem was solved by creating a new auxiliary file (an `.mdx` file) which lists the `.idx` files to be processed. A special qualifier to the `IdxTeX` command is used to specify that Master Index processing is to be performed.
- For the Master Index to be useful, it is necessary that the formatted index include *labels* which identify the volumes from which the page references come. The `.mdx` file is the obvious place for these labels to be defined. As noted above, a pointer to these label strings is included as part of the PGNODE structure, so that the labels may be easily included in the output.

Output Format Since the internal data structures contain all of the information needed to generate the Index, creation of the output file is a simple matter. All that is necessary is to walk the linked lists, generating appropriate LATEX code as we go.

The most interesting problem which occurs during output generation concerns the headings which precede the index entries which begin with a new letter of the alphabet. In the first version of the program, the heading for a new letter could appear at the bottom of one column, with the first entry for that letter appearing at the top of the next column. This was clearly undesirable.

To solve this problem, I defined a new LaTeX indexing command

```
\makeatletter
\def\indexhead#1#2#3{\par\if@nobreak \everypar{}
   \else\addpenalty{\@secpenalty}\addvspace{#1}\fi
   \begingroup #3\par \endgroup  \@xsect{#2}}
\makeatother
```

IdxTeX includes this code at the beginning of every output file it generates.

This macro was derived from LaTeX's section processing logic, where the same type of orphan problem exists. The first parameter is the amount of space to leave before the heading. The second parameter is the amount of space to leave after the heading. The third parameter is the text used to generate the heading. This macro uses `\nobreak`, `\everypar`, and `\clubpenalty` to produce the desired effect.

Summary

The IdxTeX program has been used to generate indexes in a substantial number of documents at Monsanto. We have found that its indexes are effective and attractive — well in keeping with the general quality of the documents in which they appear. It has no difficulty handling large indexes — in fact, I estimate that a document containing 25,000 `\index` commands should be well within the virtual page quotas normally found on VAX/VMS systems optimized for scientific computing environments.

While IdxTeX has basically met its design goals, a simple change in the LaTeX document styles (which was beyond the scope of this project) would allow it to do even more. At the beginning of the start of the Index to *The TeXbook*, for example, there are several paragraphs of one-column text which describe how to use the Index. The current definition of `\begin{theindex}` precludes this type of usage. I believe that a simple change to the definition of this environment (taking advantage of the optional argument of the `\twocolumn` command) would contribute to even better, more effective indexes.

Of course, IdxTeX fails to deal with the most difficult part of building an Index that communicates effectively — it does not insert the `\index` commands in the document. I leave that part of the problem to the AI experts.