Cahiers GUTenberg

AN INTERNATIONAL VERSION OF MAKEINDEX

Joachim SCHROD

Cahiers GUTenberg, nº 10-11 (1991), p. 81-90.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1991___10-11_81_0>

An International Version of *MakeIndex*

Joachim SCHROD

Technical University of Darmstadt, Institut für Theoretische Informatik
Alexanderstraße 10, W-6100 Darmstadt, FR Germany
Email: xitijsch@ddathd21.bitnet

Abstract.
MakeIndex is a powerful, portable, index processor which may be used with several formatters. But it is only usable for English texts; non-English texts – especially with non-Latin alphabets, like Russian, Arabic, or Chinese – may not easily be worked on. The tagging of index entries is often tedious and errorprone: If a markup is used within the index key, an explicit sort key must be given. A new version of *MakeIndex* is presented which allows the automatic creation of sort keys from index keys by user specified mappings. This new version does support documents in non-Latin alphabets. Furthermore it needs less main memory than the former one, and may now be used for large indexes even on small computers.

Résumé. MakeIndex *est un logiciel d'indexation puissant et portable, utilisable par plusieurs formatteurs. Mais il n'est réellement utilisable que pour les textes anglais, car il n'est pas très facile de lui faire traiter les textes écrits dans une autre langue – plus spécialement avec des alphabets non-latins, comme le russe, l'arabe ou le chinois. Le placement des entrées de l'index est souvent pénible et inducteur d'erreurs : si une entrée de l'index utilise un code de balisage il faut explicitement donner une clef de tri. Cet article présente une nouvelle version de* MakeIndex *qui permet la création automatique des clefs de tri en utilisant les règles de correspondance définies par l'utilisateur. Cette version résoud le problème des alphabets non-latins. Utilisant moins de mémoire que la version précédente, elle peut donc traiter de plus gros index, même sur des petites machines.*

Key words: index, non-english index generation, international usage of TEX, MakeIndex.

1. The Existing System

One of the most important things in a well written document is a good index. One of the most tedious work to do is the creation of a good index. Since the research on the automatic generation of indexes did not yield software systems for the general usage, the index entries must still be marked by

humans. But the processing of these entries: Association with page numbers, sorting, and producing a final form, may be done by computers.

For the scope of this paper we assume the usage of a markup system with tags by the author. (For a classification of markup systems see Coombs *et al.* [2].) Furthermore we assume that there is an interpreter for the tags available which produces a typeset, i.e. formatted, document. Examples of such systems are TEX, `troff`, Scribe, and SGML with an associated formatter. Then the index markup may be – and often is – incorporated into the usual text markup. The formatter may then output the *raw index*, i.e., the information on which pages the index entries are placed. The raw index may be sorted, polished, and transformed to a *tagged index* by an *index processor*. The tagged index will be an input for the formatter again thus producing the final index.

MakeIndex [1] is such an index processor. It was written by Pehong CHEN and Michael A. HARRISON, later on the portability was improved by Nelson BEEBE. *MakeIndex* is not bound to any formatter, it may be adapted to different systems. This adaption is done in a so-called *index style file*. There a user may specify how the syntax of the raw index file will be and what tags should be used for the output.

The transformation from raw to tagged index done by *MakeIndex* is not only a sorting process: It involves merging of page numbers for an index entry and subentry handling. At the end of the run all entries are grouped in *letter groups* containing all entries that start with a given letter. Entries starting with a non-letter are put into a special group.

MakeIndex was created with the intention "to build a complete system by analyzing the tasks involved in index processing" [1]. Well, they almost succeeded – for authors writing English documents.

2. The Problem

Of course the index markup should be as convenient as possible: It should allow other markups within the index entries, i.e., for the creation of symbols and logos, or for the specification fonts to be used for the later formatting. It should not be necessary to specify the sort order for markups anew for each index entry. Such markups fed into *MakeIndex* are quite often not exotic symbols but just national letters, like German umlauts, the French cedille, etc. Such national letters may be input as non-ASCII (or non-EBCDIC)

letters, too. The user expects them to be properly ordered as his language needs it.

MakeIndex does not know anything of markups within entries or of national letters. It handles this problem with a workaround: It allows to attach a *sort key* to an index entry. This sort key is used for sorting; merging is controlled by the printed index entry. So an author can specify where its index entry should be placed. Of course this method is error-prone: One may easily forget the sort key or may misspell it. The author might even not consider that he must specify a sort key, his national letters are nothing special to him. (In the case of national letters – say coded in ISO Latin-1 – the sort order is completely machine-dependent and may not be relied on.)

Or the index entry is not produced by hand after all; other (higher level) markups may output entries as it is senseful. Furthermore specification of a sort key may clutter the input unnecessary if there is no special structured editor available which may hide the index entries. This is a point which is often underestimated: An input must be readable per se. It will be changed, and a structured and readable input helps to find the places where the change shall happen.

But for the usage of *MakeIndex* for non-English documents problems arise even with this workaround. It's obvious for languages that don't use Latin characters, like Russian, Chinese, or Arabic. There the author must specify a sort key for every index entry, which is of course not acceptable. But even with other European languages it's problematic: The sort order differs (e.g., where are the digits to be placed?), and is sometimes not consistent within one language. An example for this are the German umlauts. The standard DIN 5007 states that they are sorted like the corresponding vowel with a following 'e'. I.e., 'ä' is sorted like 'ae'. But in phone books the umlaut is just sorted like the vowel, i.e., 'ä' is sorted like 'ae'. This leads to the case that even for languages like German or French a sort key must be specified for almost every index entry.

As this was (and is) not usable, Andreas BROSIG added a patch to *MakeIndex*, implementing the German DIN sort order. Now we could have add a lot of other patches, for every language anew, but this was not desireable. Needed is a generalized approach that enables the easy specification of different sort orders and the usage of arbitrary markups within index entries.

Still unsolved was the problem that the creation of letter groups is language dependent, as every language (or even every document) might decide in an other way what a "letter" is. Non-English languages might want to add other letters – sometimes specified as markups – which shall make up a new letter group.

An other problem does not address the functionality. *MakeIndex* uses too much main memory. Therefore a medium sized index may not be processed on systems with little memory. But a lot of authors use IBM PCs running MS-DOS and have only 640 KB available there. So the usage of *MakeIndex* was discouraged for a wide – if not for the widest – range of potential users.

Besides the problems explained above there is another inconvenience: One is not able to share parts of index styles. That lowers the reusability enourmously. E.g., there is a part of the style which describes the basic TEX markups and another part which describes the markups for a special macro package. If one skips to another macro package one has to rewrite the complete style, an automated inclusion of work already done is not possible.

3. A First Approach for an International *MakeIndex*

After analyzing the problems outlined in the previous section I decided to enhance *MakeIndex* to become an international version. Such a new version would not support multi-linguality, i.e., it assumes that the language is the same for all indexes processed in one run. A multi-lingual *MakeIndex* would not be of much use: Foreign words written in the same alphabet should be sorted in the order of the base language. How words written in an other alphabet are to be sorted is a problem of its own, the transliteration must often be chosen on a per document basis. If a document has really different parts written in different languages than it's best to create an index for every part. Of course, an international *MakeIndex* may be used for each of those indexes.

It was clear from the beginning that the first version would not be the final one. It was just created for checking the methods which I will describe below. So only the problems of different languages and markups within one index entry were addressed. All other problems were postponed.

The idea in principle is simple: An index entry consists of the *index key* and the *explicit sort key*, the last may be missing. Sorting is done by the

sort key, which is either the explicit sort key – if available – or which is generated from the index key. This generation is done by a *key mapping*. In the former *MakeIndex* this mapping was just the identity, in the new version the mapping may be specified by the user in the index style file.

The mapping is just a description how parts of an index key are transformed to parts of the created sort key. The part of the index key is specified by a UNIX-style regular expression (often called *pattern* or *regexp*). The created part of the sort key is a string with up to nine variable parts, parenthesized subexpressions from the index key pattern are inserted there. If there are several patterns matching on a part of the index key the longest one is taken. If there are several of the same length the one specified first in the index style file is taken.

As an example consider the mapping

```
\"([aouAOU])   ↦   \1e      (1)
  \['`^](.)    ↦   \1       (2)
   \([{}$])    ↦   \1       (3)
        {}$    ↦            (4)
\accent␣"..␣   ↦            (5)
```

It's a part of a mapping for the TEX typesetting system with the Plain macro package for German texts with some French words in between. The line numbers are only given for references, they are not part of the mapping specification. Line 1 specifies that a German umlaut (input as \" and the corresponding vowel) will be sorted as specified by DIN, i.e., as the vowel (the first parenthesized subexpression in the pattern) with a following 'e'. Line 2 ignores all French accents. Line 4 specifies that the TEX grouping symbols and math shift symbol are to be ignored. But this is wrong if they are to be typeset, therefore we add line 3. The order of that two lines would not matter as the rule given in line 3 is longer than those of line 4, and would therefore take precedence anyhow. Line 5 ignores all accents which were not output properly to the raw index by the macro package.

I hope that this gives an impression how powerful such a specification might be – at least at first glance.

A basic sort function was defined: The printable ASCII characters are divided into two classes, letters and other characters. Letters are sorted case insensitive. The other characters are sorted according to the ASCII order. Two special characters are available in addition, `\b` and `\e`. The first one is a

character which comes before all other characters in the alphabet, the second comes after. These new characters are needed for inserting new characters on arbitrary places.

With these concepts realized we made first positive results. But we were not satisfied:

- Non-ASCII characters were still not supported. This is especially bad with the usage of TEX 3 which now supports such characters.
- The above approach addresses only two of the outlined problems, different languages and markups inside index entries. But to be honest it solves only the last; different languages are only supported in so far as this problem may be tackled by markups.
- One could not specify how letter groups are to be built. This is especially bad for languages based on non-Latin alphabets, like Russian, Greece, or Chinese.
- What if the generated sort key, i.e., the result of the key mapping contains a national letter or a symbol. Its translation to ASCII characters must be specified immediately and might not be postponed until later. This means, e.g., that if a generated sort key should contain a German umlaut the chosen sort order must be known during the creation of the sort key. One cannot just create the German umlaut and look later how to sort this national letter.
- A similar problem arises because the author may still specify an explicit sort key. If this sort key should contain a German umlaut, how should it be input? The convenient way would be its markup form or just as a non-ASCII letter. The existing (and unconvenient) way was the input in its converted form – but the author often did not know what the converted form will be!
- Furthermore an open problem was still the merging of several index entries. What is the criterion to decide if they are equal? The index key or the sort key? The usual way was the index key. But the creation of two index entries containing the same index key, tagged in two different ways, is possible and sometimes not under the control of the user. This happens if the index is created partly automatically and partly "by hand."

4. International *MakeIndex*, Revisited

The mapping between index keys and generated sort keys is clearly too inflexible. A better way is to split this mapping into two mappings: The first one handles all imbedded markups in index keys and the second is responsible for the correct sort order.

So let's summarize the concept: A *merge key* is generated from the index entry. Either it is the explicit sort key given by the user (if such does exist) or it is mapped from the index key. This mapping is called the *merge mapping*. Afterwards the merge key is transformed into the *final sort key* by the *sort mapping*. This final sort key consists solely of printable ASCII characters and of the characters `\b` and `\e`. Sorting on these characters is done strictly according to the ASCII coding. On the result letter groups are identified by prefixes. Each prefix resembles a letter (or a letter class) in an alphabet.

As the name *merge key* implies, the decision if two index entries are the same is now based on this key instead on the index key. If two index entries with two different index keys will have the same merge key it is undefined what index key will be used for the tagged index.

Both mappings, the merge and the sort mapping, are described in the index style file by rules like those explained in the previous section. A rule consists of a keyword (`merge_rule` for the merge mapping and `sort_rule` for the sort mapping), a string with the pattern which describes on what index keys this rule should be used, and a string describing how the resulting key should be. In the description of the last string up to nine subexpressions from the pattern may be inserted.

A merge key may be mutable or immutable. If it is mutable the merge mapping will be applied again thus yielding a new merge key. The user who has specified the mapping is responsible that this evaluation will not result in an endless loop. An immutable merge key is not transformed any further. By default rules in the merge mapping produce immutable keys, transformation rules must be specially marked to create a mutable key. The mark is an asterisk preceding the output string.

Please note that the user is responsible for the fact that non-ASCII characters are not allowed in the final sort key. I.e., they must all be filtered out by the sort mapping.

A letter group is identified by a common prefix. This prefix is announced by a rule in the index style file consisting of the keyword `sort_group`, a

pattern describing the prefix, and a group number. If this group number does not equal zero then group heads may be output for this group. Usually a group head consists of the prefix determining this group. This might be changed by a rule consisting of the keyword `group head`, the group number, and a string which shall be taken as the group head. If group heads are output to the tagged index after all, is controlled by the flag `headings_flag`.

It must be emphasized that two incompatibilities to the former *MakeIndex* version were introduced: (1) Merging is now controlled by the merge key and not by the index key. (2) An index style file is now needed in any case as the pure ASCII sort order is of no use. The first incompatibility is not so important as different index entries shall have different merge keys – they would be sorted the same otherwise. The second is more a point of convenience as standard index styles are available. Style files may include each other and are searched by an environment variable (or something similar) if the operating system provides such. The most tedious thing is that the author of an index style must remember that he should usually include a standard style which lays down the basics for the chosen formatting system and the used language.

5. Fine Tuning and Cleanup Actions

When the implementation was finished we tested my approach with some "real-life examples": The biggest ones were an index for a manual on a DVI driver family, an index for a LaTeX tutorial, and an index for a course on concrete mathematics. Especially the last one showed the reasonability: There were index (sub-)entries consisting of formulas – but we needed neither an explicit sort key in any index entry nor any special rule which was just introduced for only one index entry.

But we discovered two big disadvantages: Our new *MakeIndex* was much slower (by a factor of 3) and it needed more main memory (about 50%). The increasing memory requirements were at least unsatisfying, the old *MakeIndex* was already too large for small systems without virtual memory. So *MakeIndex* needed some fine tuning.

Looking at the code we discovered that for every index entry two keys were stored in a fixed length array. Besides the wasted space it set an limit for the maximum length of an index. Now each string is allocated seperately

with just the space it needs. Furthermore each string is only stored once; many entries may point to the same string. This reduced the amount of memory for large indexes by magnitudes. Now one may process them on small DOS PCs, too.

Simple, but common, cases in the mappings (e.g., patterns without variable parts) are now treated specially. The linear search for a keyword of the index style was replaced by a hash table with a perfect hash function.[1] The style file scanner was completely rewritten; it is now only a third of size, the code structure is cleaner, and it is faster.

All changes resulted in a final version which needs less memory than the original one, and is at least as fast. (The performance depends on the amount of rules for the mappings.)

Since we had changed large parts of the system, we had to tackle the portability again. We decided to throw away all parts which substitute statements according to the used operating system respectively the used compiler. Instead we added a configuration file which describes the features of the target system and this may be used for the implementation later on. The configuration file is based on Larry WALL's `Configure` script; there exists already a lot of such configurations for other software systems which will be a good start for a new target system. This resulted in a noticable degree of code size.

6. Availability and Distribution

The resulting is system is called *MakeIndex*, Version 3.0. It is free software and available under the conditions of the GNU General Public License. Commercial companies should notice that they may sell it; they only must provide the source to their costumers (and their costumers can give it away for free). At the time of the conference *MakeIndex* 3.0 will be available "from all good TEX archives."

Now, at the time of writing, *MakeIndex* is tested on different UNIX systems (BSD, System V, AIX, and XENIX flavours), MS-DOS PCs, and VAX VMS. Test for EBCDIC mainframes will be done at the time of the conference. Other machines will be supported then, too.

[1] This function is generated by `gperf`, the perfect hash function generator of the GNU project. This program is available freely. So new keywords may be still introduced easily, we do not depend on proprietary software.

7. Open Problems and Future Works

Although we have tackled quite a few problems successfully, *MakeIndex* 3.0 is far from perfect. Just to name some open problems still waiting for volunteers: The scanner for the raw index file should be rewritten completely (I don't want to offend the author of this piece of the source – but it's flawky and bad written.) Perhaps it would be nice to allow for different types of index entries within the raw index. This would help to create different indexes out of the raw index (e.g., a person and a subject index, or indexes with two different languages) and would lower the amount of created/needed files for one TEX run.[2]

Acknowledgments

First of all I want to thank Gabor HERR who did most of the coding. He implemented all my specifications very quick and very carefully. The first positive feedback on my first ideas I got at a discussion at EuroTEX90 in Cork which pushed me forward starting the work. Klaus GUNTERMANN made a lot of helpful comments. And last, but not least, Christine DETIG was there to discuss whole evenings on the possibilities not outlined in this articles; "possibilities" which did not last longer than an evening.

References

[1] Pehong CHEN and Michael A. HARRISON, "Index preparation and processing", *Software: Practice and Experience*, Vol. 19, No. 9, September 1988, pp. 897–915.

[2] James COOMBS, Allen RENEAR, and Steven DEROSE, "Markup systems and the future of scholarly text processing", *Communications of the ACM*, Vol. 30, No. 11, November 1989, pp. 933–947.

[2] Please note the latter argument is a non-technical issue (for the system the amount of files should not matter) – but a lot of "plain users" are irritated if a bunch of files are created, and they do not know what good the files are for. For a survey of the used files see my article "The Components of TEX," published in Baskerville and in TEXline.