Xindy revisited: Multi-lingual index creation for the UTF-8 age

Joachim Schrod
Net & Publication Consultance GmbH
Kranichweg 1
63322 Rödermark, Germany
jschrod (at) acm dot org
http://www.xindy.org/

Abstract

Xindy is a flexible index processor for multi-lingual index creation. It handles 44 languages with several variants out of the box. In addition, some indexes demand special sort orders for names, locations or different target audiences; xindy can handle them as well. Raw index files may have several encodings beyond ASCII. In particular, LaTeX's standard output encoding is supported directly, as is XeTeX's UTF-8 output. With the new *xindy TUG30 release*, support for Windows is added; previously xindy was available only for GNU/Linux, Mac OS X, and other Unix-like systems.

1 What is xindy?

Xindy is an index processor. Just like MakeIndex, it transforms raw index information into a sorted index, made available as document text with markup that may be processed by TeX to produce typeset book indexes. Unlike MakeIndex, it is multi-lingual and supports UTF-8 encoding, both in the raw index input and in the tagged document output.

Overall, xindy has five key features:

1. *Internationalization* is the most important feature at all and was originally xindy's raison d'être: the standard distribution knows how to handle many languages and dialects correctly out of the box.
2. *Markup normalization* and *encoding support* is the ability to handle markup in the index keys in a transparent and consistent way, as well as different encodings.
3. *Modular configuration* enables the reusability of index configurations. For standard indexing tasks, LaTeX users do not have to do much except use available modules.
4. *Location references* go beyond page numbers. Locations may also be book names, section numbers or names, URLs, etc.
5. *Highly configurable markup* is another cornerstone. While this is usually not as important for LaTeX users, it comes in handy if one works with other author systems.

The focus of this paper is the current state of multi-lingual and encoding support that's available for xindy. The paper's scope does not include other features which I'll mention just in passing:

Locations are more than page numbers: Most index processors can work only with numbers, or maybe sequences of numbers such as "2.12". Going further, xindy features a generalized notion of structured location references that can be book names, music piece names, law paragraphs, URLs and other references. You can index "Genesis 3:16" and "Exodus 3:16" and Genesis will be in front of Exodus since they are not alphabetically sorted names any more, but terms in an enumeration.

Such location references may be combined into a range, such as 6, 7, 8, and 9 becoming "6–9". Also well-known are range specifications in the humanities, such as 6f or 6ff. With xindy, location ranges can also be formed over structured references, but some knowledge about the domain of the reference components must be available.

xindy is configured with a declarative style language, where declarations look like

```
(some-clause argument1 argument2 ...)
```

A file with such declarations is called an xindy module, and an xindy run may use several of these modules. This allows making available predefined modules for common indexing tasks, e.g., the f- and ff-range designation illustrated above. Xindy declarations are also used to configure output markup.

Last, but not least, xindy is the practical result in research about a theoretical model of index creation. Even if one does not use xindy the program, the model itself can provide valuable input for the creation of future index creation programs.

2 Multi-lingual sorting

Sorting is a multi-layered process where characters are first determined, placed into categories that are

albanian	greek	norwegian
belarusian	gypsy	polish
bulgarian	hausa	portuguese
croatian	hebrew	romanian
czech	hungarian	russian
danish	icelandic	serbian
dutch	italian	slovak
english	klingon	slovenian
esperanto	kurdish	spanish
estonian	latin	swedish
finnish	latvian	turkish
french	lithuanian	ukrainian
general	lower-sorbian	upper-sorbian
georgian	macedonian	vietnamese
german	mongolian	

Table 1: Predefined languages in xindy

sorted the same for now (collation classes), and sorted either left-to-right or right-to-left. If this results in index entries that are sorted the same but are not identical, characters are reclassified with different rules and sorted again to resolve the ambiguities. Words from most languages can be sorted with this process. It is standardized as ISO/IEC 14651 (*International String Ordering*).

2.1 Predefined languages

Xindy provides the ability to sort indexes in different languages; 44 of them are already prepackaged in the distribution and are listed in Table 1. For some of these languages there are multiple sorting definitions: e.g., German has two different kind of sort orders, colloquially called DIN and Duden sorting (more on that later).

While the sorting of all predefined languages may be expressed in terms of the ISO standard 14651 named above, xindy's abilities go beyond that. The standard language modules are usable for indexes where index entries all belong to one language or where foreign terms are sorted as if they would be local. But if one mixes several languages in one index, e.g., in an author index, one is able to define the sort rules that should be used individually, just for this text. While this is some work, of course, xindy at least makes it possible to create indexes for such real international works that go beyond mere multi-lingualism.

2.2 Complexity of index sorting

One might ask if this paper doesn't make a mountain out of a molehill, and what's the big deal with all this supposed complexity of index sorting and creation

To address that valid question, I'd like to present a few peculiarities that show why index sorting is more complex than just sorting a few strings and why an ISO standard on string ordering is a good start but not the end of ordering index entries.

Cultural peculiarities For some languages, index sort order depends on context, or the term's semantics. German is a good example for this complexity: there are two sort orders in wide usage and they differ in the sorting the "umlauts". These are the vowels with two dots above: ä, ö, and ü.

One sort order sorts them as if there were a following 'e', i.e., 'ä' is sorted as 'ae', 'ö' is sorted as 'oe', and 'ü' is sorted as 'ue'. That is the official sort order, and is defined in an official German standard, DIN 5007. This sort order is used for indexes in publications for the domestic market, for an audience that knows German and is thus expected to know that these characters are true letters on their own and not just vowels with accents. Such a domestic audience is also expected to have learned the sort order of umlauts in their first school year and will be able to cope with that cultural peculiarity.

A second way of index sorting drops that idiosyncratic German feature and sorts umlauts as if they were vowels with accents, i.e., 'ä' is sorted just like 'a', and so on. This sort order is used in indexes of publications for an international market, or where an international audience is expected to read this publication regularly. Especially dictionaries and phone books use this non-standard way of sorting; we want to give our foreign visitors a chance to look up the phone number of any Mr. or Ms. Müller they want to visit. This sort order has no official name, but is colloquially known as phone book sorting or "Duden sorting", after the most important dictionary of the German language that uses this sort order.

Legacy rules Some special and non-obvious sort orders are so old that the reason behind them is not known (at least, not to me). An interesting example is French, where additional complexity has been introduced in some previous time when it comes to sorting names with accented characters: when two words have the same letters but differ in accents, the existence of accents decides the final sort order — but backwards, from right to left!

The most prominent example is the four words

cote côte coté côté

In the first pass of sorting a French index, these four words are sorted the same. In the second pass, they are still sorted the same — the second pass sorts uppercase letters before lowercase letters. The third pass then sorts from right to left and puts

non-accented letters before accented letters:

⟶
cote
⟵
c**ô**te
cot**é**
c**ôté**

This finally results in the sort order shown in the table above.

Character recognition Sometimes legacy representations in files on computers introduce complexities: In Spanish, 'ch' and 'll' are *one* letter and sorted accordingly, whereas in all other European languages they are two letters. Traditionally, these Spanish letters are represented by two characters in a file; an index sort processor has to recognize them as such.

There is also the problem of what to do when a character appears in an index that does not exist in that language, e.g., there is no 'w' in Latin. Should one be pragmatic and sort it like modern languages of the Roman family, between 'v' and 'x'? Or should one place it somewhere in the non-letter group? The order might very well depend on the target audience and intent of the respective index.

In the TEX world, character recognition is arguably less an issue for non-Latin scripts — TEX authors are used to specifying their characters with exact encodings or transliteration. Especially the rise of Unicode text editors and their enhanced input support for non-Latin scripts make identification of characters easier than the supposedly ASCII-like representation in traditional encodings.

Beyond Europe The examples so far were "just" about European languages. (Admittedly, because I know most about them . . .) Some languages use phonetic sorting where one needs additional information about words that are used in the sort algorithm. This does not change the algorithm itself, but available authoring systems often do not support that aspect at all. (xindy does not support it out-of-the-box either, but it has the functionality to describe such sorting in its language modules.)

Other languages use aspects of glyphs such as strokes or number of strokes for sorting. Diacritics may or may not influence sorting; sometimes they are vowels, sometimes they just denote special emphasis and can be ignored.

3 One sort order is not enough

For multi-lingual index creation it is not sufficient to define sort orders for languages. Having defined a language module with the default sort order of German, French, or any other language is a good and necessary start, and many index processors stop at that. But it is not sufficient for production of actual indexes where sorting rules appear that are not covered by standards.

For example, in author indexes some languages handle parts of nobility names differently, depending on whether they are part of the name or a true peerage title. In registers of places, city names might be sorted differently than spelled. Transliteration must be taken into account, just like combination of alphabets within one index.

This boils down to the requirement that project- or document-specific sort rules are needed. While one book might sort 'St.' as it is written, Gault Millau will need a different sort order for its register — they sort it as 'Saint'. MakeIndex introduced a way to do that by specifying print and sort keys explicitly, as in `\index{Saint Malo@St. Malo}`: It demands from the author that this explicit specification must be used every time that term appears.

Xindy goes a step further and allows the user to specify sort orders in a separate style file that may be used just for one document or reused for all documents in a project. It still allows using an explicit sort and print key in your TEX document — but experience has shown that it is much less error-prone to declare it once in an external file for a whole group of index terms than to write it explicitly in each occurrence of that group.

4 Examples for xindy style declarations

Let's have a look at how such document-specific declarations are done. We demonstrate their use for two purposes: index entry normalization and entry sorting.

Markup normalization is the process to decide if two raw index entries denote the same term and should be combined into one processed index entry, i.e., if they should be *merged*.

Especially with TEX, it might be that the same term appears differently in the raw index. This is caused by macro expansions, especially when one produces part of the index entries automatically. Depending on the state of TEX's processing, macros in the raw index are sometimes expanded and sometimes not expanded.

Here comes into play a point where xindy differs from MakeIndex: it ignores TEX or LATEX tags (macro names and braces) by default. With xindy, you can write `\index{\textbf{term}}` and this will be the index entry "term", `\textbf` and the braces will be ignored. (Such index entries are usually not

input manually, but are generated by other macros.)

So, how would one index METAFONT, written in the Metafont logo font? The MakeIndex way would be to use `\index{METAFONT@\MF}` for every to-be-indexed occurrence of METAFONT, and that way still works with xindy. But in addition, one can use an xindy style file with the declaration

```
(merge-rule "\MF" "METAFONT")
```

and just use `\index{\MF}` in the document. No risk to add typos to one's index entries; these index entries will be sorted as 'METAFONT' but output as METAFONT.

Merge rules may influence whole index terms or just parts of them. One can also use regular expressions to normalize large classes of raw terms.

Sort specifications Sort orders are specified with very similar declarations:

```
(sort-rule "ä" "a~e")
```

tells xindy to place 'ä' between 'a' and 'b'. (The special notion '`~e`' means "at the end"; there is also '`~b`' for "at beginning".)

This is the low-level way to specify sort orders, and it is used to create special document- or project-specific sort orders as mentioned above. It is possible to create whole language sort-order modules with that method as well — we did so at the start of the xindy project.

But then Thomas Henlich wrote *make-rules*, a preprocessor to create xindy language modules for languages where sort order can be expressed with the ISO 14651 concept. For that preprocessor, one describes alphabets and sort orders over collation classes with multiple passes, and xindy modules with sort rules as shown above are created as a result.

5 Encoding of raw index files — LICR and UTF-8

At the moment, the most often used encoding for raw index files is the LaTeX output of `\index` commands. That encodes non-ASCII characters as macros; the representation is called *LaTeX Internal Character Representation* or LICR, as described in section 7.11 of *The LaTeX Companion*, 2nd ed. Xindy knows about LICR: xindy modules exist with merge rules to recognize these character representations. A special invocation command for LaTeX, `texindy`, picks them up automatically, so authors have no need to think about them.

At the moment, LICR is mapped to an ISO-8859 encoding that's appropriate for the given language, and that encoding is then the base for xindy's sort rules. Please note that this is completely unrelated to the encoding used in the author's LaTeX document. You can use UTF-8 there with the inputenc package, but that encoding doesn't matter for index sorting. When the raw index arrives at xindy, that original encoding is not visible any more; we see only LICR. And we just need ISO-8859 encodings for sorting languages that are supported by LaTeX's standard setup, which mostly use European scripts.

While this is appropriate and useful for European languages, it won't help authors with documents in Arabic, Hebrew, Asian, or African languages. But they also won't use LICR much anyhow and will probably be better served by new programs like XeTeX or Omega/Aleph. For these users, all language modules are supplied in a variant that knows about UTF-8 encodings as output by XeTeX or Omega's low-level output of (Unicode) characters. If one has a raw index file that was produced by these systems, one can use xindy; it will "just work".

Looking beyond UTF-8 is still not necessary in the TeX world; we have no TeX engine that will output UTF-16 or even wider characters to a raw index file or expect such encodings in a processed index file. That's good, because xindy can't handle UTF-16 input — yet. This will probably be an enhancement of one the next major releases and shall help to open up xindy's appeal beyond TeX-based authoring environments.

6 Availability

Through release 2.3, xindy was available only for GNU/Linux and other Unix-like systems. At the time of writing, release 2.4 has been prepared which is nicknamed the *TUG30 release*, to honor TUG's 30th birthday. This release adds support for Windows (2000, XP, and Vista), thus widening the potential user base considerably. For now, installation of a Perl system is needed to use xindy; this should not be a big obstacle.

The TUG30 release is available for download at xindy's Web site `www.xindy.org`. Currently, it is there in source form; binary distributions for several operating systems will be added as time permits.

While release 2.3 is included in TeX Live 2008, with executables for nearly all the platforms except Windows, including Mac OS X, we were not able to finish the new release 2.4 in time to make it to the DVD. Hopefully, it will become available via the new on-line update mechanism soon. Eventually, full support for xindy will be available in TeX Live 2009.

The best place to look for user documentation about xindy is *The LaTeX Companion*, 2nd ed., section 11.3. Documentation on the Web site is technically more complete, but improvements of its organization and accessibility are high on our to-do list.