Testing indexes: testidx.sty

Nicola L. C. Talbot

Abstract

The testidx package [10] provides a simple method of generating a test document with a multi-paged index for testing purposes. The dummy text and index produced is designed to replicate problems commonly encountered in real documents.

The words and phrases indexed cover the basic Latin set A(a), ..., Z(z) and some extended Latin characters, such as Ø(ø), Æ(æ), Œ(œ), Å(å), Þ(þ) and Ł(ł), to test the indexing application's ability to sort according to various Latin alphabets (such as Swedish or Icelandic). Version 1.1 also includes some words starting with digraphs, Dd(dd), Dz(dz), Ff(ff), IJ(ij), Ll(ll), Ly(ly), Ng(ng), and a trigraph Dzs(dzs), to test alphabets where these are considered separate letters (such as Welsh, Dutch or Hungarian).

There are also some numbers and symbols indexed that don't have a natural word order.

1 Introduction

There are a number of problems that can occur when generating an index using LaTeX. These may relate to the index style (`\printindex`), the way the indexing information is written to an external file (`\index`) or the way the indexing application (such as `xindy` or `makeindex`) performs. A large document may have a complicated and slow build process, which can be frustrating when making minor adjustments to the index layout. The testidx package provides a way to create a test document that can be used to enhance the required style. Section 5 shows how the sample text can be extended to include tests for other languages or scripts.

The simplest test document is:

```
\documentclass{article}
\usepackage{makeidx}
\usepackage{testidx}
\makeindex
\begin{document}
\testidx
\printindex
\end{document}
```

Version 1.1 of testidx comes with the supplementary package testidx-glossaries, which uses the interface provided by the glossaries package [9] instead of testing `\index` and `\printindex`. In this case, the simplest test document is:

```
\documentclass{article}
\usepackage{testidx-glossaries}
\tstidxmakegloss
\begin{document}
\testidx
\tstidxprintglossaries
\end{document}
```

2 Intentional issues

The dummy text is designed to introduce issues that your style or build process may need to guard against. These allow you to test the document style, the way the indexing information is written to the external file, and the way the indexing application processes that information.

2.1 Stylistic issues

The style issues are those which need addressing through the use of LaTeX code within the document itself, or in the class or package that deals with the index style, or within a style file or module used by the indexing application which controls the LaTeX code that's written to the output file. The test document should load the appropriate document class and indexing package to match your real document.

2.1.1 Page breaking

There are enough entries for the index to span multiple pages. If you have letter group headings in your index style there's a good chance that there will be at least one instance of a page or column break occurring between a heading and the first entry of that letter group. There's also a chance that a break will also occur between a main entry and the first of its sub-entries.

This does, of course, depend on the font and page dimensions. You may need to adjust the geometry to cause an unwanted break before experimenting with adjusting the style to prohibit it.

2.1.2 Headers and footers

Since the index spans multiple pages, it's possible to test the headers and footers for the first page of the index as well as subsequent even and odd pages. This is useful if the header or footer content needs to vary and you need to check that this is done correctly.

2.1.3 Line breaking

The index contains a mixture of single words, compound words, phrases, names, places and titles. This means that some of the entries are quite wide, which can cause line breaking problems in narrow columns.

2.1.4 Whatsits

Some of the entries are indexed immediately before the term, for example

```
\index{page break}page break
```

and some are indexed immediately after the term, for example

```
paragraph\index{paragraph}
```

The whatsit introduced by `\index` can cause problems. This is most noticeable in an example equation where the indexing interferes with the limits of a summation. In practice, the `\index` would need to be moved to a more suitable location, but the example provided by the dummy text helps to highlight the problem.

2.2 Index recording issues

The way that indexing typically works is to write the entry data (using `\index`) to an external file that's then input and processed by the indexing application. This write operation can sometimes go wrong causing incorrect information to be written to the external file. (There's no test for incorrect syntax within the argument of `\index`. It's assumed you know how to correctly index entries. The tests here are for the underlying operation of `\index`.)

The glossaries package uses a similar method but instead of using `\index`, the file write instruction is internally performed by commands like `\gls` and `\glsadd`.

2.2.1 Page breaking

The dummy text has some long paragraphs with indexing scattered about them. This increases the chance of a page break occurring mid-paragraph (although again it depends on the font and page dimensions). TEX's asynchronous output routine can cause page numbers to go awry, and this provides a useful way to check that the page number is written correctly to the external file.

2.2.2 Extended Latin characters

The indexed entries include terms that contain non-ASCII letters (either through accent commands like `\'` or using UTF-8 characters). The UTF-8 encoding isn't an issue for the modern XeLaTeX or LuaLaTeX engines, but it is a problem for the older LaTeX formats. If your engine doesn't natively support UTF-8 and you have characters outside the basic Latin set, then this is something that needs to be tested. The testidx package has four modes to test this, depending on your set-up.

1. ASCII with LaTeX commands stripped. ('Bare ASCII')

   This mode is triggered through the use of LaTeX with testidx's `stripaccents` option and without the inputenc package [2]. This option is the default, so the first test example above is in this mode. This mode emulates doing, for example:

   ```
   \index{elite@\'elite}
   ```

   So if this is the way that you're indexing words in your real document, this is the mode you need in the test document.
2. Unmodified ASCII. ('ASCII Accents')

   This mode is triggered by running LaTeX with testidx's `nostripaccents` option and without the inputenc package. This mode emulates doing

   ```
   \index{\'elite@\'elite}
   ```

   Use this mode if in your real document you are simply doing, for example, `\index{\'elite}`.
3. Active UTF-8.

   This mode is triggered if the inputenc package with the `utf8` option is loaded and testidx is loaded afterwards with the `nosanitize` option. This emulates doing

   ```
   \index{élite}
   ```

   Since the inputenc package makes the first octet of a UTF-8 character active, this causes the entry to be expanded as it's written to the index file, so that it appears as:

   ```
   \IeC {\'e}lite
   ```

   Use this mode if in your real document you are doing, for example, `\index{élite}` and you want to test how it's written to the index file. (This mode is the default when using XeLaTeX or LuaLaTeX, but the characters aren't active, so it's much the same as the next mode.)
4. Sanitized UTF-8.

   The three modes listed above are for emulating different `\index` usage. This last mode really belongs in the next section as it's provided for testing the indexing application's UTF-8 support, but is included in this list for completeness.

   This mode is triggered if inputenc is loaded with the `utf8` option and testidx is loaded afterwards with the `sanitize` option. This emulates doing

   ```
   \def\word{élite}\@onelevel@sanitize\word
   \expandafter\index\expandafter{\word}
   ```

   The sanitization isn't applied to any remaining content of `\index`, such as the encap. For example,

   ```
   \index{ð|see{eth (ð)}}
   ```

   is implemented such that only the ð part before the encap is sanitized so this would end up written to the index file as

   ```
   ð|see{eth (\IeC {\dh })}
   ```

   (testidx doesn't modify the `\index` command, but uses the `\expandafter` approach where the control sequence has a combination of sanitized and non-sanitized content.)

There's no support for other encodings.

2.3 Indexing application issues

An indexing application typically reads the external file created by LaTeX (the *input file* that contains data, discussed in the previous section) and produces another file (the *output file* that contains typesetting instructions) which can then be read by LaTeX (through commands like `\printindex`). The terminology here is a little confusing as the input file from the indexing application's point of view is an output file from LaTeX's point of view and vice versa. For consistency, the indexing application's point of view is used here.

The dummy entries are designed to test the indexing application's ability to collate entries into an ordered list where each entry has an associated set of page references (locations) or cross-references. The list may be sub-divided into letter groups, according to the initial letter of each entry. The definition of a 'letter' depends on the collation rule. For example, 'aeroplane', 'Ängelholm', 'Ångstrom̈' and 'Aßlar' may all belong to the 'A' letter group according to one rule (such as English) but may belong to different letter groups according to another rule (such as Swedish). In some languages, a 'letter' may actually be a digraph (such as 'dz') or a trigraph (such as 'dzs'). Entries that don't belong in any of the recognised letter groups are typically put into a default or 'symbols' group.

2.3.1 Extended Latin characters, digraphs and trigraphs

As mentioned above, the test entries include some words with extended Latin characters, digraphs and a trigraph to test the localisation support of the indexing application used in the document build process. There are three digraphs (ll, ij and dz) that may instead be represented by a single UTF-8 glyph (ǈ, ij and ǆ). The `diglyphs` option will switch to using these glyphs instead, but remember that the document font must support those characters if you want to try this.

2.3.2 Collation-level homographs

The words 'resume' and 'résumé' are both indexed. These should be treated as separate entries even if the comparator used by the indexing application considers them identical. Check that both words appear in the index. Similarly for index/`\index` and recover/re-cover.

2.3.3 Compound words

The test entries include space- or hyphen-separated compound words to test the sort rule. Different rules have different ways of treating spaces or hyphens. One rule may ignore those characters (for example, 'vice-president' < 'viceroy' < 'vice versa') whereas another rule may treat a space as coming before a hyphen (for example, 'vice versa' < 'vice-president' < 'viceroy').

2.3.4 Numbers

The test entries include some numbers (2, 10, 16, 42, 100). The indexing application may identify these as numbers and order them numerically, or it may simply order them as a sequence of non-alphabetical characters (so 2 would be placed after 100).

2.3.5 Symbols

The test entries include two types of symbol entries. The first set are mathematical symbols, such as α (`\alpha`). The second set are the markers used in the dummy text to indicate where the indexing is taking place. The package options `prefix` and `noprefix` determine how these entries are indexed.

The `prefix` option (default) inserts the character `>` before the sort value for mathematical symbols and inserts the character `<` before the sort value for the markers. For example:

```
\index{>alpha@$\alpha$}
```

for α and

```
\index
{<tstidxmarker@\csname tstidxmarker\endcsname
\space (\tstidxcsfmt {tstidxmarker})}
```

for the symbol · produced by testidx's marker command `\tstidxmarker`. This naturally gathers the two types of symbols. A sophisticated indexing application may then be customized to treat the character `>` as the 'maths' letter group and `<` as the 'marker' letter group.

The `noprefix` option doesn't insert these characters. This emulates simply doing

```
\index{alpha@$\alpha$}
```

for α (which puts α in the 'A' letter group) and

```
\index
{tstidxmarker@\csname tstidxmarker\endcsname
\space (\tstidxcsfmt {tstidxmarker})}
```

for the marker (which puts this symbol in the 'T' letter group).

A real document will likely provide syntactic commands for this type of indexing. For example, to index a maths symbol that's produced using a single control sequence (such as `\alpha`):

```
\newcommand{\indexmsym}[1]{%
 \index{#1@$\csname #1\endcsname$}}
```

The symbol is then indexed as, for example,

```
\indexmsym{alpha}
```

The `prefix` option simply emulates a minor adjustment to such a command to alter the sorting.

There are additional maths symbols that aren't governed by the prefix options as they start with alphabetical characters. These are simply indexed in the form:

```
\index{f(x)@$f(\protect\vec{x})$}
```

so they end up in the associated letter group ('F' in the above example).

There are also terms starting with a hyphen (command line switches) to test sorting. For example:

```
\index{-l (makeindex)@\protect
\tstidxappoptfmt{-l}
(\protect\tstidxappfmt{makeindex}}
```

These again aren't affected by the prefix options as the hyphen forms part of the term. Conversely, there are some terms starting with a backslash that have the leading backslash omitted from the sort term. For example

```
\index{index@\protect\tstidxcsfmt{index}}
```

2.3.6 Multiple encaps

There are three test commands, which simply change the text colour, used as page encapsulator (encap) values. One of the dummy blocks of text has the same word ('paragraph') indexed multiple times with different encap values. For example, no encap:

```
\index{paragraph}
```

The first test encap (`\tstidxencapi`):

```
\index{paragraph|tstidxencapi}
```

Similarly for the second (`\tstidxencapii`) and third (`\tstidxencapiii`) test encaps. If all instances occur on the same page then this causes an encap clash for that entry on that page. The indexing application may or may not have a method for dealing with this situation.

2.3.7 Inconsistent encap in a range

There are some explicit ranges formed using `(` and `)` at the start of the encap value. For example, block 4 of the dummy text includes

```
\index{range|(}
```

which is closed in block 9 with

```
\index{range|)}
```

However in block 5, this term is indexed with one of the test encaps:

```
\index{range|tstidxencapi}
```

This can't be naturally merged into the range and causes an inconsistency. The indexing application may or may not have a method for dealing with this.

2.3.8 Cross-referenced terms

Some terms are considered a synonym of another term. Instead of duplicating the location lists for both terms, it's simpler for one term to redirect to the other in an index. This is typically done with the `see` encap. For example:

```
\index{gobbledegook|see{gibberish}}
```

The dummy text, like a real world document, will only index this type of term once so it only has one location which is encapsulated by `\see{`⟨*other word*⟩`}{`⟨*page*⟩`}`. Since this command ignores the second argument, no actual location will be visible in the page list.

The other type of cross-reference is done with the `seealso` encap (which has the same syntax as `see`). For example

```
\index{padding|seealso{filler}}
```

These types of entries will be indexed in other places as well to create a location list that has both page references and the cross-referenced term. In some cases (as in the above example) the encap's argument exactly matches the referenced term, but in other cases it doesn't. This inconsistency may or may not cause a problem for the indexing application.

One term in particular that's tested needs checking. The word 'lyuk' is first indexed without an encap, then indexed with the `seealso` encap and later indexed again without an encap. If the indexing application simply treats the `seealso` encap as just another formatting command, this can end up with the rather odd occurrence of the cross-reference appearing in the middle of the location list.

2.3.9 Untidy page lists

Some of the entries are indexed sporadically throughout the dummy text. Depending on the font size and page dimensions, this could result in a sequence of consecutive page numbers that can be concatenated into a neat range or it could lead to an untidy list that has odd gaps that prevent a range formation.

3 testidx-glossaries

The supplementary testidx-glossaries package loads testidx and glossaries. The commands used in the dummy text are altered to use `\glsadd` or `\gls`. The dummy entries all need to be first defined and the indexing activated. This is done with

```
\tstidxmakegloss
```

The glossary is then displayed with

```
\tstidxprintglossary
```

or with

```
\tstidxprintglossaries
```

(which will display all defined glossaries using the analogous command).

There are some minor differences in the package options shared by both testidx and testidx-glossaries, and there are some supplementary options only available with testidx-glossaries:

extra Load the extension package glossaries-extra [8].

nodesc Each entry is defined with an empty description (default). The `mcolindexgroup` style is set. You can override this in the usual way. For example:

```
\setglossarystyle{mcolindexspannav}
```

desc Each entry is defined with a description. In this case, the `indexgroup` style is set, but again you can override it.

makeindex This option is passed to glossaries and ensures that `\tstidxmakegloss` uses

```
\makeglossaries
```

and `\tstidxprintglossary` uses

```
\printglossary
```

The indexing should be done by `makeindex`, invoked directly or via the `makeglossaries` Perl script or the `makeglossaries-lite` Lua script.

xindy This option is passed to glossaries and again ensures that `\tstidxmakegloss` uses

```
\makeglossaries
```

and `\tstidxprintglossary` uses

```
\printglossary
```

The indexing should be performed by `xindy` (again either invoked directly or through one of the provided scripts).

tex This ensures that `\tstidxmakegloss` uses

```
\makenoidxglossaries
```

and `\tstidxprintglossary` uses

```
\printnoidxglossary
```

The indexing is performed by TEX and is *slow* — the document build may appear as though it has hung.

bib2gls This implicitly specifies `extra` and also passes the `record` option to the glossaries-extra package. In this case, `\tstidxmakegloss` uses

`\GlsXtrLoadResources[`⟨*options*⟩`]`

and `\tstidxprintglossary` uses

```
\printunsrtglossary
```

In this case, the indexing should be performed by `bib2gls` [7], a Java command line application designed to work with glossaries-extra. The ⟨*options*⟩ and the number of instances of `\GlsXtrLoadResources` varies according to the package settings (such as `prefix` or `diglyphs`). More detail is provided later on (see page 391).

manual Use this option if you don't want to use the helper commands `\tstidxmakegloss` and `\tstidxprintglossary`. You will need to ensure you pass the appropriate options to the glossaries or glossaries-extra package and load the files containing the entry definitions.

4 Examples

The following examples can be used to test the various indexing methods. To compile them, you need to have at least testidx version 1.1. For the examples using testidx-glossaries, it's best to have at least version 4.30 of glossaries and version 1.16 of glossaries-extra.

The letter groups created by each example are shown in Table 1 (in the order they appear in the index). In the table, 'Symbols' indicates the symbols group (which in `xindy` parlance is the default group), 'Numbers' indicates the group containing numerical terms and 'Other' indicates a headless group beyond the end of the alphabet. Some of the examples create their own custom groups. If a group contains initial letters that may not be expected to appear in that group (such as accented versions) then those letters are included afterwards in parentheses.

The contents of the symbols group for each example are shown in Table 2, where 'markers' indicates the marker commands prefixed with `<`, 'maths' indicates the mathematical symbols prefixed with `>`, 'switches' indicates the terms starting with a hyphen, 'non-ASCII' indicates the terms where the sort value starts with a non-letter ASCII character (typically the backslash `\` at the start of accent or ligature commands, such as `\'` or `\oe`) and 'UTF-8' indicates the terms where the sort value starts with a UTF-8 character that doesn't fall into any of the recognised letter groups, according to the indexer's alphabet.

The ordering of the switches is shown in Table 3, and the ordering of the mathematical symbols is shown in Table 4 with the corresponding sort values shown in parentheses. These may all be in the symbols group or in their own group or scattered throughout the index in the various letter groups, as indicated in Table 1.

The ordering of the numbers (which may or may not be in their own group) is shown in Table 5, the collation-level homographs in Table 6, and a selection of compound words in Table 7.

The place name Aßlar contains an eszett (ß). In the bare ASCII mode this is indexed as

```
\index{Asslar@A\ss lar}
```

while in the ASCII accents mode it's indexed as

```
\index{A\ss lar}
```

and in UTF-8 mode it's indexed as

```
\index{Aßlar}
```

Although Aßlar always appears in the 'A' group, its location within that group varies, as shown in Table 8.

Further tables show location lists:

- Table 9: for the entry with multiple encaps ('paragraph', Section 2.3.6);
- Table 10: for the entry with the explicit range interruption ('range', Section 2.3.7);
- Table 11: for the entry with the mid-`seealso` encap ('lyuk', Section 2.3.8);
- Table 12: for an entry with a ragged page list ('block', Section 2.3.9).

A shell script was created for each example with the build process so that the complete document build could be timed (using the Unix `time` command). The elapsed real time ⟨*minutes*⟩:⟨*seconds*⟩ for each example is shown in Tables 13 for testidx and 14 for testidx-glossaries.

▸ **Example 1** (`makeindex` and bare ASCII mode)
This builds on the example shown earlier with a `makeindex` style file to enable the group headings, the fontenc package [4] to provide the commands `\dh` (ð), `\th` (þ) and `\TH` (Þ), and the amssymb package [5] to provide the spin-weighted partial derivative `\eth` (ð). These extra packages allow for more test entries that would otherwise be omitted.

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage{filecontents}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
\usepackage{makeidx}
\usepackage{testidx}
\makeindex
\begin{filecontents}{\jobname.ist}
headings_flag 1
heading_prefix "\\heading{"
heading_suffix "}\n"
\end{filecontents}
\newcommand{\heading}[1]{%
 \item\textbf{#1}\indexspace}

\begin{document}
\testidx
\printindex
\end{document}
```

This uses the default settings `prefix` and (since there's no UTF-8 support) `stripaccents`. The heading command is simplistic as these examples are testing the indexing applications rather than the index style. The build process is:

```
pdflatex doc
makeindex -s doc.ist doc
pdflatex doc
```

(where the file is called `doc.tex`).

The terms that are placed in the alphabetical groups have been ordered using a case-insensitive word comparator, the numbers have been sorted numerically (Table 5) and the symbols have been sorted using a case-sensitive comparator, as can be seen by the ordering of the switches (Table 3). Since the accent commands have been stripped, the words are all placed in the basic Latin letter groups (Table 1).

▸ **Example 2** (`makeindex` and ASCII accents mode)
This is the same as the previous example except for the package option:

```
\usepackage[nostripaccents]{testidx}
```

This doesn't strip the accents so, for example, 'élite' is indexed as `\'elite`. This causes all the words starting with extended Latin characters to appear in the symbols group (Table 2) due to the leading backslash in the control sequences. Since `\AA` expands to `\r A`, Å ends up between œ (`\oe`) and þ (`\th`). 'Aßlar' is placed at the start of the 'A' letter group before 'aardvark' (Table 8) since the second character in the sort key is a backslash (from the start of `\ss`) which comes before 'a'.

▸ **Example 3** (`makeindex`, bare ASCII mode and no prefixes)
This is the same as Example 1 except for the package option:

```
\usepackage[noprefix]{testidx}
```

This doesn't insert the `<` and `>` prefixes that kept the markers and maths together in Example 1. The markers remain close to each other as they still start with the same sub-string (now `tstidx` instead of `<tstidx`) but they have been moved to the 'T' letter group. The maths symbols are now scattered about the index (Table 1), for example, α is in the 'A' letter group (since its sort value is now `alpha`). Only the switches remain in the symbols group (Table 2).

▸ **Example 4** (`makeindex`, ASCII accents mode and no prefixes)
This is the same as Example 2 except for the extra package option:

```
\usepackage[nostripaccents,noprefix]{testidx}
```

As with Example 3, the marker and maths entries are no longer in the symbols group (Table 1), but as with Example 2 that group (Table 2) now contains the terms starting with accent commands (as well as the switches).

▸ **Example 5** (`makeindex -l`)
This is the same as Example 1 except for the build process which uses `makeindex`'s `-l` switch:

```
pdflatex doc
makeindex -l -s doc.ist doc
pdflatex doc
```

This changes the ordering of the compound words shown in Table 7 (except for 'yo-yo'). The ordering is still case-insensitive for words (Table 8) and case-sensitive for symbols (Table 3).

▸ **Example 6** (`makeindex` and sanitized UTF-8)
This is like Example 1 but UTF-8 support has been enabled through the inputenc package:

```
\usepackage[utf8]{inputenc}
```

The default `sanitize` option is on, which means that the UTF-8 characters in the sort key are sanitized and so don't expand when writing the input file. The build process used in Example 1 fails because `makeindex` isn't configured for UTF-8 and the resulting output file is corrupt. This can almost be fixed with `iconv` except near the end of the file, which triggers the error

```
\heading{iconv: illegal input sequence
```

This is because only the first octet (C3) of a two-octet character has been put in the argument of `\heading`. The only way to avoid this is to omit the headings, so the build process is:

```
pdflatex doc
makeindex -o doc.tmp doc
iconv -f utf8 doc.tmp > doc.ind
pdflatex doc
```

The 'Other' groups shown in Table 1 highlight the way that `makeindex` is sorting according to each octet, so the first group after Z contains Á (C3 81), Ä (C3 84), Å (C3 85), Í (C3 8D), Ö (C3 96), Ø (C3 98), Ú (C3 9A), Þ (C3 9E), æ (C3 A6), é (C3 A9), ð (C3 B0) and þ (C3 BE). From `makeindex`'s point of view, these all belong to the C3 letter group (which is why it tried to write the character C3 as the argument of `\heading` when the headings setting was on).

The next few examples use `xindy` to perform the indexing. The `makeindex` style file (`.ist`) is no longer applicable. An `xindy` module (`.xdy`) is used instead. A straight substitution of `makeindex` with `texindy` causes an error message with the sample entries:

```
ERROR: Cross-reference-target
("\\tstidxstyfmt {inputenc}") does not exist!
```

Unlike `makeindex`, `texindy` recognises the `see` and `seealso` encaps as cross-references (rather than just a formatting command). This error is the result of

```
\index{fontencpackage@\tstidxstyfmt {fontenc}
package|seealso{\tstidxstyfmt {inputenc}}}
```

('fontenc package, *see also* inputenc'). `texindy` checks that the cross-referenced term also exists, but there's no exact match here as the cross-referenced term was indexed slightly differently using

```
\index{inputenc package@\tstidxstyfmt
{inputenc} package}
```

('inputenc package'). This inconsistency is the result of a stylistic choice to avoid the repetition of the word 'package' in the exact match 'fontenc package, *see also* inputenc package'.

If you want to ignore these kinds of inconsistencies, you can switch off the automatic verification in the `.xdy` file when defining a cross-reference class. For example:

```
(define-crossref-class "seealso"
 :unverified)
```

Unfortunately with `texindy` this causes the error

```
ERROR: replacing location-reference-class
`"seealso"' is not allowed !
```

since the `seealso` class has already been defined (in the file `makeindex.xdy`, which is loaded by `texindy` to provide compatibility with `makeindex`). One possible workaround is to define a custom module and use `xindy` directly (instead of using `texindy`).

In your real document you can circumvent this issue by ensuring an exact match in your `see` and `seealso` encap arguments or by writing your own custom `xindy` module that defines the `seealso` class as unverified.

Alternatively, you can create your own custom cross-reference encap. For example

```
(define-crossref-class "uncheckedseealso"
 :unverified)
(markup-crossref-list
 :class "uncheckedseealso"
 :open "\seealso" :close "{}")
```

and use this instead. The testidx package allows you to try this out by providing a command to set your own cross-reference encap value. For example:

```
\tstidxSetSeeAlsoEncap{uncheckedseealso}
```

The problematic cross-reference now becomes

```
\index{fontencpackage@\tstidxstyfmt{fontenc}
package|uncheckedseealso{\tstidxstyfmt
{inputenc}}}
```

which uses `uncheckedseealso` instead of `seealso`.

The examples below circumvent this issue by using `xindy` directly with a custom module.

▸ **Example 7** (`xindy` and sanitized UTF-8)
The sample `xindy` style provided here mostly replicates `texindy.xdy` but doesn't load `makeindex.xdy`.

The cross-reference classes (`see` and `seealso`) both have the verification check switched off. This custom module also has to define the location classes provided by `makeindex.xdy` and define the test encap values used by testidx.

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage{filecontents}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{amssymb}
\usepackage{makeidx}
\usepackage{testidx}

\makeindex

\begin{filecontents*}{\jobname.xdy}
(require "latex.xdy")
(require "latex-loc-fmts.xdy")
(require "latin-lettergroups.xdy")

(define-crossref-class "see" :unverified)
(markup-crossref-list :class "see"
 :open "\see{" :sep "; " :close "}{}")

(define-crossref-class "seealso" :unverified)
(markup-crossref-list :class "seealso"
 :open "\seealso{" :sep "; "
 :close "}{}")

(markup-crossref-layer-list :sep ", ")

(define-location-class-order
 ("roman-page-numbers"
  "arabic-page-numbers"
  "alpha-page-numbers"
  "Roman-page-numbers"
  "Alpha-page-numbers"
  "see"
  "seealso"))

; list of allowed attributes
(define-attributes ((
 "tstidxencapi"
 "tstidxencapii"
 "tstidxencapiii" )))

; define format to use for locations
(markup-locref :open "\tstidxencapi{"
 :close "}" :attr "tstidxencapi")
(markup-locref :open "\tstidxencapii{"
 :close "}" :attr "tstidxencapii")
(markup-locref :open "\tstidxencapiii{"
 :close "}" :attr "tstidxencapiii")

; location list separators
(markup-locref-list :sep ", ")
(markup-range :sep "--")

\end{filecontents*}

\begin{document}
\testidx
\printindex
\end{document}
```

The build process is

```
pdflatex doc
xindy -M doc -L english -C utf8 -t doc.ilg \
      doc.idx
pdflatex doc
```

The ordering for some of the extended characters is a little odd with the `english` setting. For example, ß comes between 'n' and 'p' (Table 8) and Á, Ä, Å, Í and Ú are all in the O letter group (Table 1). They have not been considered either symbols (like Ć, which doesn't occur in English words) or sorted according to their base letter (like é, which does). Better results are obtained with the language set to `general`, which is used later in Example 17.

The switches aren't placed in the symbols group but have instead been placed in the alphabetical letter groups (ignoring the initial hyphen). The numbers (which are now in the symbols group) have been sorted as strings rather than numerically (Table 5).

The term `\index` is present, but the word 'index' has been omitted (Table 6) and its page list has been merged with the `\index` locations. A real world document would need to ensure unique sort keys. (For example, use `index.cs` as the sort value for `\index`.) The other collation-level homographs 'recover'/'re-cover' and 'resume'/'résumé' don't have this problem as the sort values for each pair are non-identical even though the comparator may consider them equivalent.

▸ **Example 8** (`xindy`, sanitized UTF-8 and letter order)
The sorting in Example 7 can be adjusted to letter ordering by adding the following line to the custom `.xdy` file:

```
(require "letter-order.xdy")
```

This alters the ordering of the compound words (see Table 7), but this doesn't quite match the order produced by `makeindex`'s letter order option used in Example 5 for the hyphenated words. The terms '`\index`' and 'index' have again been merged due to their identical sort values (Table 6), and the switches are in the alphabetical letter groups (Table 1) but their locations within those groups have changed as a result of the spaces being ignored.

▸ **Example 9** (`xindy`, sanitized UTF-8 and ignore hyphen)
The previous example can be slightly altered by changing `letter-order` to `ignore-hyphen`. There's no difference here from Example 8 in the order of the collation-level homographs 'recover' and 're-cover' (Table 6). There is a difference in the ordering of the compound words shown in Table 7, which is back to the word order from Example 7, and the switches are

still in the alphabetical groups, so there's no noticeable difference between this example and Example 7. It seems that `xindy` always ignores hyphens regardless of whether or not the `ignore-hyphen` module is loaded.

▸ **Example 10** (`xindy`, sanitized UTF-8 and ignore punctuation)
Another option is to use the `ignore-punctuation` module. However, swapping `ignore-hyphen` in the previous example for `ignore-punctuation` causes an error while reading `ignore-punctuation.xdy`:

```
#<OUTPUT STRING-OUTPUT-STREAM>> ends within
a token after multiple escape character
```

The problem seems to come from the line

```
(sort-rule "\"" "")
```

If I remove

```
(require "ignore-punctuation.xdy")
```

and replace it with the contents of that file without the problematic line, the document is able to compile.

This example differs from the previous one, as it also causes the prefix characters `<` and `>` to be ignored, so this behaves much like the `noprefix` option with the maths and markers placed in the alphabetical letter groups (Table 1).

The sorting is still case-insensitive, but the difference caused by the ignored punctuation can be seen in the ordering of the switches. For example, the term `-l (makeindex)` is now treated as `lmakeindex` (all punctuation stripped) instead of `l(makeindex)` (only hyphen and space stripped), so it's now after `-L icelandic (xindy)` (since 'i' < 'm') whereas in the previous example it came before `-L danish (xindy)` ('(' < 'd').

▸ **Example 11** (`xindy`, sanitized UTF-8 and numeric sort)
Example 7 can be easily modified to sort the numbers numerically by adding the line:

```
(require "numeric-sort.xdy")
```

to the start of the `.xdy` file. A separate group for the numbers can also be defined in this file:

```
(define-letter-group "Numbers"
 :prefixes ("0" "1" "2" "3" "4" "5" "6"
  "7" "8" "9") :before "A")
```

The ordering of the defined attributes tells `xindy` the order of precedence when there's an encap clash (see Section 2.3.6). In the previous example, the `tstidxencapi` encap took precedence in the conflict in the 'paragraph' entry (see Table 9), but there are still two instances of page 2 in the location list as the default encap (where no encap has been specified) has been kept as well as the dominant `tstidxencapi` encap. This can be fixed by adding `default` to the end of the list of allowed attributes:

```
(define-attributes ((
"tstidxencapi" "tstidxencapii"
"tstidxencapiii" "default")))
```

This will cause a warning

```
WARNING: ignoring redefinition of
attribute "default" in
(DEFINE-ATTRIBUTES
 (("tstidxencapi" "tstidxencapii"
"tstidxencapiii" "default")))
```

This is because `latex-loc-fmts.xdy` already contains an attribute list containing `default`:

```
(define-attributes (("default" "textbf"
 "textit" "hyperpage")))
```

To remove the warning, delete the line

```
(require "latex-loc-fmts.xdy")
```

from the custom `.xdy` file. Any of the usual LaTeX attributes, such as `hyperpage`, that are provided in the file `latex-loc-fmts.xdy` can be added to the custom attributes list if required.

▸ **Example 12** (`xindy`, sanitized UTF-8, no prefixes and numeric sort)
The previous example is modified here so that it doesn't use the `>` and `<` prefixes. The testidx package is now loaded using:

```
\usepackage[noprefix]{testidx}
```

inputenc is again loaded to enable UTF-8 support. The markers and maths symbols are now placed in the letter groups (Table 1). For example, α now has the sort value `alpha`, so it's in the A letter group, and ∂ has the sort value `partial`, so it's in the P letter group.

▸ **Example 13** (`xindy`, active UTF-8 and numeric sort)
Example 11 is modified here so that it doesn't sanitize the sort value. The testidx package is now loaded using:

```
\usepackage[nosanitize]{testidx}
```

The inputenc package is again loaded to enable UTF-8 support, which means that the first octets of the UTF-8 characters are active so they are expanded when written to the index file. This causes the `xindy` error

```
ERROR: CHAR: index 0 should be less than
the length of the string
```

This error occurs when the sort value is empty.

Recall from Example 7 that the example's custom module loads the file `latex.xdy`. This in turn loads `tex.xdy` which strips commands and braces

from the sort key. This means that the sort keys that solely consist of commands (such as `\IeC{\TH}`) collapse to an empty string, which triggers this error.

As a result of the error, no output file is created, so the document doesn't contain an index. One way to force this example document to have an index is to remove the line

```
(require "latex.xdy")
```

and add the content of `latex.xdy` without the line

```
(require "tex.xdy")
```

but this means that all the words starting with extended characters end up in the symbols group since the initial backslash in `\IeC` is a symbol (which is what we'd get if we use `makeindex` instead).

An alternative approach is to keep `latex.xdy` and add a merge rule for the problematic entries:

```
(merge-rule "\\TH *" "TH" :eregexp :again)
(merge-rule "\\th *" "th" :eregexp :again)
```

(and similarly for other commands like `\ss` and `\dh`) before loading `latex.xdy`.

This example uses this simpler method, which strips all the `\IeC` commands but converts the commands (such as `\TH`) representing characters. This essentially reduces the sort values to much the same as the bare ASCII mode in Example 1. In both this example and Example 1, the sort value for 'résumé' becomes 'resume'. This means that two distinct terms have identical sort values. In `makeindex`'s case, the terms are deemed separate entries as the actual part is different, but `xindy` merges entries with identical sort values, so only one of these two terms ('résumé') appears in the index. (As happens with 'index' and '`\index`', and again it's the first term to be indexed that takes precedence.)

The alphabetical ordering is now reasonable for English, but not for other languages, such as Swedish or Icelandic, that have extended characters, such as ø or þ, that form their own letter groups. (This wouldn't change even if the language option specified with `-L` changes as there are no actual extended characters in the index file, just control sequences representing them.)

This example provides a useful illustration between using TEX engines that natively support UTF-8 and simply enabling UTF-8 support through inputenc. Replacing inputenc and fontenc with fontspec and switching to XeLaTeX or LuaLaTeX shows a noticeable difference. It's therefore not enough to have a Unicode-aware indexing application, but it's also necessary to ensure the extended characters are correctly written to the indexer's input file.

▸ **Example 14** (`xindy`, sanitized UTF-8, custom groups and numeric sort)
This example returns to using the `sanitize` option so that the UTF-8 characters appear correctly in the index file. We can build on Example 11 to create two custom groups that recognise the < and > prefixes:

```
(define-letter-group "Maths"
 :prefixes (">") :before "Numbers")
(define-letter-group "Markers"
 :prefixes ("<") :before "Maths")
```

I also tried to define a similar group for the switches:

```
(define-letter-group "Switches"
 :prefixes ("-"))
```

but this doesn't work (Table 1) as the hyphen is by default ignored (see Example 9). Setting a sort rule for the hyphen doesn't seem to make a difference.

Now the default symbols group (Table 2) only contains the UTF-8 characters that aren't recognised by the language module.

▸ **Example 15** (`xindy -L icelandic`, sanitized UTF-8, custom groups and numeric sort)
It's time to try out some other languages. This example uses the same document and style from Example 14 but substitutes `icelandic` for `english` in the `xindy` call. This results in some extra letter groups (see Table 1).

The Icelandic alphabet has ten extra letters (in addition to the basic Latin set) Á(á), Ð(ð), É(é), Í(í), Ó(ó), Ú(ú), Ý(ý), Þ(þ), Æ(æ) and Ö(ö). There is a letter group for the ð entry, but it's headed with the lower case ð rather than the upper case Ð. (All the other letter groups are headed with an upper case character, including Þ.) There are also letter groups for Þ, Æ and Ö, but not for the acute accents.

The non-native characters have a more logical ordering than in the English examples with ß treated as 'ss' (Table 8), but Ä and œ are in the Æ group (Table 1) and Ø is in the Ö letter group. The symbols group contains the remaining extended characters (Table 2).

▸ **Example 16** (`xindy -L hungarian`, sanitized UTF-8, custom groups and numeric sort)
As above but now using `-L hungarian`. This also results in some extra letter groups (such as Ö), but there are some missing groups that should be in the Hungarian alphabet, such as the digraphs Dz(dz) and Ly(ly), and the trigraph Dzs(dzs).

The O letter group contains an odd collection of extended characters, such as Ä, Å, Þ and ð. As with the `english` setting, ß has an unexpected location between 'n' and 'p' (Table 8).

In theory it should be possible to add letter groups for digraphs and trigraphs using a similar method as the other custom groups:

```
(define-letter-group "Dz"
 :prefixes ("DZ" "Dz" "dz")
 :after "D" :before "E")
```

Unfortunately this doesn't work as the 'D' letter group takes precedence because it was defined first. (The language modules are loaded before the custom module.) A complete new language module is needed to make this work correctly, which is beyond the scope of this article. Another possibility is to use glyphs instead of the digraphs, but this is only possible for digraphs that have a glyph alternative.

▸ **Example 17** (`xindy`, sanitized UTF-8, selected digraph glyphs, custom groups and numeric sort) This example is like Example 14 but the `diglyphs` option is used.

```
\usepackage[diglyphs]{testidx}
```

This means that instead of using the two characters 'dz' in words like 'dzéta', the single glyph dz is used. It should now be possible to create the Dz letter group as in the example above but with the glyphs DZ, Dz and dz.

```
(define-letter-group "ʣ"
 :prefixes ("ʣ" "ʣ" "ʤ")
 :after "D" :before "E")
```

Similarly for IJ, ij and IL, ɬ. There's no glyph used in the trigraph dzs.

Since these characters are not easily supported by inputenc and fontenc, it's necessary to use XeLaTeX or LuaLaTeX instead. This means replacing inputenc and fontenc with fontspec.

```
\usepackage{fontspec}
```

Some fonts don't support these glyphs (ij is the most commonly supported of this set), so the choice here is quite limited. Some fonts support the glyphs in only one family or weight. For example, Linux Libertine O and FreeSerif support all glyphs in the default medium weight but the ɬ and IL glyphs are missing from bold. I've chosen DejaVu Serif for the document font in this example as it has the best support of all my available fonts:

```
\setmainfont{DejaVu Serif}
```

The change in font slightly alters some of the page lists in the index. The build process is now:

```
xelatex doc
xindy -M doc -L general -C utf8 \
      -t doc.ilg doc.idx
xelatex doc
```

(I've set the language to `general` to reflect the mixture of alphabets.)

This example generates a warning from `xindy`:

```
WARNING: Found a :close-range in the
index that wasn't opened before!
Location-reference is 5 in keyword (range)
I'll continue and ignore this.
```

The altered page breaking caused by the font change has resulted in both the opening range produced with

```
\index{range|(}
```

and the interrupting encap produced with

```
\index{range|tstidxencapi}
```

to occur on page 2. The open range encap is dropped in favour of the `tstidxencapi` encap. This means that the closing range

```
\index{range|)}
```

on page 5 no longer has a matching opening range, so no range is formed (Table 10).

As can be seen from Table 1, there's no symbols group for this example. The markers and maths have been assigned to their own groups through the use of their < and > prefixes, the numbers are in their own number group, the glyphs dz, ij, and ɬ have been assigned to separate groups, and the remaining UTF-8 characters have all been assigned to the basic Latin letter groups, as a result of the `general` language setting. The switches still have the hyphen ignored and so are in the letter groups.

The trigraph dzs is still unrecognised, as are the dd, ff, ly and Ng digraphs, which haven't been replaced with glyphs. (As most TeX users will know, there is a glyph for the ff digraph in most fonts, but although the sequence `ff` is usually converted to a ligature when typesetting, it's written to the index file as two characters. There's no corresponding glyph for the title case version Ff.)

The examples now switch to testidx-glossaries, which provides extra sorting methods. Some of the informational blocks of text are altered by this package, so the page numbers may be different in the location lists due to the difference in some paragraph lengths.

Instead of using `\index`, the terms are first defined using

```
\newglossaryentry{⟨label⟩}{⟨options⟩}
```

where ⟨*label*⟩ (which can't contain special characters) uniquely identifies the term and ⟨*options*⟩ is a ⟨*key*⟩=⟨*value*⟩ list. The main keys are `name` (the way the term appears in the glossary) and `description`. By default the sort value is the same as the name (as `\index` when `@` isn't used) but the `sort` key can

be used to provide a different value. The files containing these definitions are automatically loaded by `\tstidxmakegloss`.

The terms are then displayed and indexed using commands like `\gls{`⟨*label*⟩`}` throughout the document text. This will display the value of the `text` key, which if omitted defaults to the same as `name`.

For example, with the normal indexing methods, the term $f(\vec{x})$ can be displayed and indexed in the text using

```
\[ f(\vec{x})\index{f(x)@$f(\vec{x})$} \]
```

whereas with glossaries the term is first defined in the preamble:

```
\newglossaryentry{fx}{name={$f(\vec{x})$},
 text={f(\vec{x})},
 sort={f(x)},
 description={}}
```

and then used in the document:

```
\[ \gls{fx} \]
```

In the text this does `f(\vec{x})` (the value of the `text` key), in the index this does `$f(\vec{x})$` (the value of the `name` key), and it's sorted by `f(x)` (the value of the `sort` key).

Cross-references are performed using the `see` key, for example:

```
\newglossaryentry{padding}{name={padding},
 see={[\seealsoname]filler},description={}}
```

(where the `see` value is a comma-separated list of labels optionally preceded by a tag) or using `\glssee`, for example,

```
\glssee[\seealsoname]{padding}{filler}
```

The glossaries-extra package provides the `seealso` key, which is essentially the same as `see` with the tag set to `\seealsoname`. If this key is detected, it will be used instead. For example:

```
\newglossaryentry{padding}{name={padding},
 seealso={filler},description={}}
```

These methods essentially index the reference as:

```
padding?\glossentry
{padding}|glsseeformat[\seealsoname]{filler}
```

with Z as the location (the glossaries package uses `?` instead of `@` as the actual character).

Since `makeindex` by default lists upper case alphabetical locations last, this automatically moves the cross-reference to the end of the list.

▸ **Example 18** (testidx-glossaries and `makeindex`) The basic test document is:

```
\documentclass{article}
\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
\usepackage{testidx-glossaries}
\tstidxmakegloss

\renewcommand*{\glstreenamefmt}[1]{#1}
\renewcommand*{\glstreegroupheaderfmt}[1]{%
 \textbf{#1}}

\begin{document}
\testidx
\tstidxprintglossaries
\end{document}
```

The `mcolindexgroup` glossary style sets the name in bold by default, so I've redefined `\glstreenamefmt` to prevent this. (There's no need to distinguish the name when there are no descriptions.)

For this example, my build process is

```
pdflatex doc
makeglossaries-lite doc
pdflatex doc
```

This uses the Lua script rather than the Perl script. The Lua script simply determines the required indexing application (in this case `makeindex`) and the correct options from the `.aux` file and runs it. The `makeglossaries` Perl script does more than this and is used in the next example.

For comparison, an explicit call to `makeindex` was also used:

```
pdflatex doc
makeindex -t doc.glg -o doc.gls -s doc.ist \
          doc.glo
pdflatex doc
```

The only difference in the result is in the build time, which is slightly faster. The times for both build methods are shown in Table 14.

Since the inputenc package isn't used, accents are stripped as with Example 1. This means it's emulating, for example:

```
\newglossaryentry{elite}{name={\'elite},
 sort={elite},description={}}
```

There are some differences between the index produced in this example and that produced in Example 1 (aside from the page numbering and the differences between the index and glossary styles). The ordering of `\index` and 'index' have changed (Table 6). In Example 1, the control sequence `\index` is indexed as

```
index@\tstidxcsfmt{index}
```

and the term 'index' is just indexed as `index`. With glossaries the control sequence is effectively indexed as

```
index?\glossentry{cs.index}
```

and the term is effectively

```
index?\glossentry{index}
```

When makeindex encounters terms with identical sort values, it seems to give precedence to terms where the sort value is identical to the actual value. So in the first example, 'index' (which has no separate sort) comes before \index. With glossaries, both have a distinct sort and actual value.

A similar thing happens with 'resume'

```
resume?\glossentry{resume}
```

and 'résumé'

```
resume?\glossentry{resumee}
```

Since the accents have been stripped, both terms have 'resume' as the sort value. (Since active characters can't be used in labels and labels must be unique, the label for the second term is resumee.)

▸ **Example 19** (testidx-glossaries and makeglossaries)
This example uses the same document as the previous one above, but uses the makeglossaries Perl script in the build process instead of the Lua script:

```
pdflatex doc
makeglossaries doc
pdflatex doc
```

The difference here can be seen in the location list for the 'paragraph' entry (see Table 9). The script has detected makeindex's multiple encap warning and tried to correct the problem. Version 2.20 incorrectly gives precedence to a non-range encap over an explicit range encap which then causes makeindex to trigger the error

```
-- Extra range opening operator (.
```

This is the same problem that occurred with xindy in Example 17. makeglossaries version 2.21 (provided with glossaries v4.30) corrects this and gives the range encaps precedence. The only problem that remains is just the inconsistent page encapsulator within a range warning.

▸ **Example 20** (testidx-glossaries, bare ASCII mode and xindy)
The test document from Example 18 can be modified to use xindy instead of makeindex by adding the xindy package option:

```
\usepackage[xindy]{testidx-glossaries}
```

The glossaries package provides a custom xindy module (automatically generated by \makeglossaries). Minor adjustments can be made before the module is written using commands or package options. For example, to add the test encaps:

```
\GlsAddXdyAttribute{tstidxencapi}
\GlsAddXdyAttribute{tstidxencapii}
\GlsAddXdyAttribute{tstidxencapiii}
```

Again we can take advantage of the < and > prefixes:

```
\GlsAddLetterGroup{Maths}{:prefixes (">")
 :before "glsnumbers"}
\GlsAddLetterGroup{Markers}{:prefixes ("<")
 :before "Maths"}
```

(The glossaries package provides its own version of the numbers group called glsnumbers.)

The numeric-sort module isn't loaded by default, so it needs to be explicitly added if numerical ordering is required:

```
\GlsAddXdyStyle{numeric-sort}
```

The above lines all need to go before

```
\tstidxmakegloss
```

The build process is:

```
pdflatex doc
makeglossaries doc
pdflatex doc
```

The Lua alternative can also be used, or a direct call to xindy:

```
xindy -L english -I xindy -M doc -o doc.gls \
      -t doc.glg doc.glo
```

The difference between this example and the earlier xindy examples is that the indexing information is written in xindy's native format, for example

```
(indexentry
 :tkey (("elite" "\\glossentry{elite}") )
 :locref "{}{3}"
 :attr "pageglsnumberformat" )
```

(pageglsnumberformat is the default encap used by glossaries in xindy mode when the format key hasn't been set and the page counter is used for the locations.)

The example document doesn't load inputenc, which means the bare ASCII mode is on, which is why the accent doesn't appear in the sort field (identified in :tkey). This means that the sort value for 'résumé' is once again 'resume' and the conflicting unaccented 'resume' is lost (Table 6). The hyphens are again ignored so the switches are placed in the alphabetical letter groups (Table 1).

▸ **Example 21** (testidx-glossaries, sanitized UTF-8 and xindy)
This example makes a minor adjustment to the previous one by adding

```
\usepackage[utf8]{inputenc}
```

This enables the sanitized UTF-8 mode so the sort values contain UTF-8 characters. (The glossaries package automatically sanitizes the sort key by default, but the testidx-glossaries package will ensure that its own nosanitize option is honoured, which just passes sanitizesort=false to glossaries.)

The build process again uses `makeglossaries`. Since the document hasn't loaded any language packages, the language option written to the `.aux` file defaults to English so `makeglossaries` calls `xindy` with `-L english`. This means the extended characters are ordered in the same way as in Example 14 (Table 1).

▸ **Example 22** (testidx-glossaries, `xindy` and non-standard page numbering)
`makeindex` can only recognise roman (i, I), arabic (1) and alphabetic (a, A) locations. `xindy` has more flexibility, so this example makes a minor adjustment to the previous example to use an unusual page number scheme. This requires etoolbox [3] (automatically loaded by glossaries) for `\newrobustcmd`, and the stix package [1] for the six dice commands `\dicei`, ..., `\dicevi`:

```
\newrobustcmd{\tally}[1]{%
 \ifnum\number#1<7
  $\csname dice\romannumeral#1\endcsname$%
 \else
  $\dicevi$%
  \expandafter\tally\expandafter{\numexpr#1-6}%
 \fi
}
\renewcommand{\thepage}{\tally{\arabic{page}}}
```

The page numbers are now represented by dice. For example, page 2 is ⚁ and page 10 is ⚅⚀.

Since the stix package by default automatically changes the document font, which will alter the page breaking, I've used the `notext` option to prevent this:

```
\usepackage[notext]{stix}
```

This allows a better comparison with the previous example.

The locations are now written to the indexing file in the form `\\tally {`⟨*n*⟩`}`, where ⟨*n*⟩ is the page number. (The backslash is automatically escaped by glossaries. The space is significant.) `xindy` needs to be informed of this new location format:

```
\GlsAddXdyLocation{tally}{
 :sep "\string\tally\space{"
 "arabic-numbers" :sep "}"}
```

Aside from the location presentation, there is one difference between this example and the previous one when used with versions of glossaries below 4.30, and that's the cross-reference location. For example, with glossaries v4.29, the 'lyuk' entry appears as '*see also* digraph, ⚂, ⚃' but for v4.30 it appears as '⚂, ⚃, *see also* digraph' (Table 11). This is due to a bug that has been corrected in v4.30.

▸ **Example 23** (testidx-glossaries, bare accents mode and TEX)
If, for some reason, you're unable or unwilling to use an external indexing application, the glossaries package provides a method of alphabetical sorting using TEX. The document from Example 18 can be adapted to use this method by adding the `tex` option:

```
\usepackage[tex]{testidx-glossaries}
```

The accents are stripped by default so the sorting is just performed on the basic Latin set.

The build process is simply

```
pdflatex doc
pdflatex doc
```

This method is considerably longer than the others (see Table 14) and has the worst results.

There's no numbers group with this method. The numbers are included with the symbols (Table 2), but are ordered numerically (Table 5). The ordering of the compound words has changed (Table 7) with somewhat eccentric results. There are no range formations, even for explicit ranges, and the range interruption (Table 10) interrupts the list formatting (a space is missing).

The 'see also' cross-reference in Table 11 doesn't interrupt the location list, but this is only because the `see` key was used when defining the entry (which is why it's at the start of the list). If `\glssee` had been used instead within the document, it would have produced the same result as Example 1.

▸ **Example 24** (testidx-glossaries, bare accents mode and TEX with letter ordering)
The previous example used the glossaries package's default `sort=standard` setting, which sets the entry `sort` key, if omitted, to the `name` key and optionally sanitizes it. The command `\printnoidxglossary` also accepts a `sort` key in the optional argument to allow different ordering for different glossaries. (This capability is not available with `\printglossary`.) The localised `sort` key allows the values `word` and `letter` for word and letter ordering, so this example replaces

```
\tstidxprintglossaries
```

with

```
\printnoidxglossary[sort=letter]
```

to test letter order sorting with TEX. This again takes a long time (Table 14). The ordering of the compound words (Table 7) now matches the `xindy` letter order in Example 8. There's a change in the order of one of the collation-level homographs from the previous example: 're-cover' is now after 'recover' (Table 6). Other than that, this method produces much the same results as the previous example.

So far the examples have all used alphabetical ordering for the majority of the entries based on the value of the `sort` key (or the `name`, if `sort` is omitted). The glossaries package also allows sorting according to definition or use. The next few examples illustrate this.

▸ **Example 25** (testidx-glossaries and order of definition with `makeindex`)
The glossaries package provides the options `sort=def` and `sort=use` to switch to order of definition or first use within the document. The code used in Example 19 needs to be adjusted to pass this option since glossaries is being loaded implicitly:

```
\PassOptionsToPackage{sort=def}{glossaries}
\usepackage{testidx-glossaries}
```

Alternatively (glossaries v4.30):

```
\usepackage{testidx-glossaries}
\setupglossaries{sort=def}
```

This method works by overriding the `sort` value so that it's just a number that is incremented every time a new entry is defined. This means that `makeindex` orders numerically, and all entries are placed in the numbers group (Table 1). It therefore makes no sense to use a style with group headings with this option. The entries that are actual numbers (Table 5) are no longer in numerical order according to their value given in the `name` field.

The build process again uses `makeglossaries`, which deals with the conflicting encaps for page 3 (Table 9). This method is faster than Example 18 (Table 14) as it's simpler to compare two integers than to perform a case-insensitive word-order comparison between two strings.

▸ **Example 26** (testidx-glossaries and order of definition with `xindy`)
This is like the previous example, but `xindy` is used:

```
\PassOptionsToPackage{sort=def}{glossaries}
\usepackage[xindy]{testidx-glossaries}
```

The attributes (encaps) need to be specified as in Example 20, but since we're sorting by order of definition it's not possible to define the maths or markers groups.

Since numeric comparisons are faster than string comparisons, the `numeric-sort` style from Example 20 is also used (Table 14). This example will still work without that style as the sort values are zero-padded to six digits. (If you have 1,000,000 or more entries, you'll need `numeric-sort` to enforce numerical comparisons.)

The glossaries package automatically defines the numbers group, so all entries are placed in that. If the package option `glsnumbers=false` is also passed to glossaries, then the entries will instead be placed in the default group.

There's no longer a problem with the collation-level homographs (Table 6) as the sort values are now unique numbers, so 'index' and 'resume' have reappeared in the index.

▸ **Example 27** (testidx-glossaries and order of definition with TEX)
This example makes a minor change to the document used in Example 24:

```
\printnoidxglossary[sort=def,nogroupskip,
style=mcolindex]
```

This orders by definition but no actual sorting is performed here. The glossaries package keeps track of which entries have been defined in an internal list associated with the glossary that contains the given entry. The entry label is appended to the list when it's defined, so the list is already in the correct order. Each time an entry is used in the document, a record is added to the `.aux` file. This also provides a list of all entries that have been indexed, which is naturally in the order required by `sort=use` (order of use). All that is needed is to iterate over the appropriate list and display each entry that has a record.

Now that TEX doesn't have to sort the entries, the build process is much faster (Table 14). The only problem here is that the style must be changed to one that doesn't use group headings, as otherwise TEX has to determine the correct heading from the sort value. Unlike the previous two examples, the sort key isn't altered to a numeric value (because `sort=def` wasn't passed as a package option). This means that a new group will be started with pretty much every entry unless the entries happen to be defined in alphabetical order. So in this example I've switched the style to `mcolindex` and used the `nogroupskip` option. The build process is the same as for Example 23.

This method has a problem with sub-entries. Unlike `makeindex` and `xindy`, there's no hierarchical sorting with this method (because there's no actual sorting) so if a sub-entry isn't defined immediately after its parent is defined then it won't appear immediately after its parent in the glossary. Furthermore, if a sub-entry is used, its parent won't automatically be indexed.

The dummy text contains a number of top-level entries that are duplicated as sub-entries. For example, the book *Ulysses* is defined as:

```
\newglossaryentry{Ulysses}
{name={\tstidxbookfmt{Ulysses}},
 sort={Ulysses},description={}
}
```

but a sub-entry is defined immediately after:

```
\newglossaryentry{books.Ulysses}
{name={\tstidxbookfmt{Ulysses}},
 parent={books},
 sort={Ulysses},description={}
}
```

These are then referenced using:

```
\gls{Ulysses}\glsadd{books.Ulysses}
```

The parent entry (`books`) hasn't been used in the dummy text, so it doesn't appear in the glossary. This leads to the rather odd result:

Ulysses 2

Ulysses 2

The first instance is the top-level entry and the second instance is the sub-entry. Even if the parent entry (`books`) had been used, it would still be separated from its sub-entry (`books.Ulysses`) as it's not defined immediately before it, but is one of the first entries to be defined.

The location ranges (Table 10) have the same problems as for Example 23, but the build time is significantly faster, although it's still slower than using `makeglossaries` (Table 14).

This method is essentially for non-hierarchical symbols that don't have a natural alphabetical order and the available build tools are somehow restricted.

The glossaries-extra package extends the base glossaries package, providing new features (such as the `category` key and associated attributes) and reimplementing existing methods (such as the abbreviation handling). This package can automatically be loaded by testidx-glossaries through the option `extra`. This also ensures that each entry is assigned a category. For example, the *Ulysses* entry is now:

```
\newglossaryentry{Ulysses}
{name={\tstidxbookfmt{Ulysses}},
 category={book},
 sort={Ulysses},description={}
}
```

(and similarly for the sub-entry). This doesn't alter the indexing, but it can be used to modify the way the entries are displayed.

▸ **Example 28** (testidx-glossaries and glossaries-extra in order of definition)

The glossaries-extra package provides another way of displaying the list of entries in order of definition. Unlike the above examples, this includes *all* entries, not just the ones that have been indexed. This is done with

`\printunsrtglossary[`⟨*options*⟩`]`

which simply iterates over all defined entries in that glossary, displaying each one in turn according to its handler, so it's similar to Example 27 but doesn't check if the term has been indexed.

This method doesn't create any external indexing files, so `\tstidxmakegloss` isn't needed in this example. The `.tex` files containing the definitions for the dummy entries can be loaded using `\input` or `\loadglsentries`, but it's simpler to just use:

```
\tstidxloadsamples
```

which means you don't have to worry about remembering the file names. However there's a problem here. The `see` key can only be used after the indexing has been initialised (through `\makeglossaries` or `\makenoidxglossaries`). This was a precautionary measure introduced because the cross-reference information can't be indexed before the associated file has been opened, and users who defined entries before using `\makeglossaries` were puzzled as to why the cross-references didn't show up. The error alerts them to the problem.

The simplest solution is to prevent the use of the `see` key in the test entries with the `noseekey` option provided by testidx-glossaries.

```
\documentclass{article}

\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
\usepackage{amssymb}
\usepackage[extra,noseekey]{testidx-glossaries}

\tstidxloadsamples

\setglossarystyle{mcolindex}
\renewcommand*{\glstreenamefmt}[1]{#1}

\begin{document}
\testidx
\printunsrtglossary[nogroupskip]
\end{document}
```

(An alternative is to pass `seenoindex=ignore` to the glossaries package or pass `autoseeindex=false` to the glossaries-extra package.) The document build process is simply:

```
pdflatex doc
```

Some terms that are used in the original dummy text provided by testidx aren't present in the slightly altered version produced by testidx-glossaries. (This is why imakeidx is missing from the glossary examples listed in Table 6.) These terms are still defined by testidx-glossaries to provide an additional test, if required, for the treatment of non-indexed

entries. Since \printunsrtglossary includes all entries, imakeidx is once again in the index even though it's not in the dummy text.

The most noticeable difference is the absence of page lists (Tables 9, 10, 12) and cross-references (Table 11). No indexing has been performed so there's no record of where the entries have been used. There are no groups (Table 1). This method suffers from the same problem as Example 27 with the sub-entries separated from their parents.

This example is faster than all the other examples using testidx-glossaries (Table 14), but the build only requires a single LaTeX call and doesn't perform any sorting, so that's hardly surprising. It's slower than Example 1 (Table 13): makeidx is a small, simple package and therefore fast to load whereas glossaries and glossaries-extra are complicated and rely on a number of other packages.

A few seconds can be shaved off the build time by adding

```
\setupglossaries{sort=none}
```

before the entries are defined. (Only available with glossaries version 4.30 onwards.) This skips the code used to set up the sort values (such as sanitizing and escaping special characters for makeindex or xindy).

The iteration handler recognises three special fields, group, location and loclist, which don't have a key provided by default. The group value should be a label identifying the letter group, and will only be checked for by the handler if the group key is defined. For example:

```
\glsaddstoragekey{group}{}{\glsgroup}
```

The location value may contain any valid code that produces the location list. Although the group field must have an associated key of the same name for the handler to recognise it, the location field can simply be set using \GlsXtrSetField.

The loclist value must be in the same format as the internal lists provided by etoolbox where each item is in the format

```
\glsnoidxdisplayloc{⟨prefix⟩}
  {⟨counter⟩}{⟨encap⟩}{⟨location⟩}
```

for locations, or

```
\glsseeformat[⟨tag⟩]{⟨label⟩}{}
```

for cross-references. (This is the same command used by makeindex and xindy when the see key is used. The final argument is the location for the benefit of makeindex but is always ignored.) The loclist value can't be provided as a key since it requires a specific separator used by etoolbox. Instead, each item can be added to the list using

```
\glsxtrfieldlistadd{⟨label⟩}{⟨field⟩}{⟨item⟩}
```

The group value must be a label (no special characters) because it's used as a hypertarget with the 'hyper' or 'nav' glossary styles. The corresponding title can be set using

```
\glsxtrsetgrouptitle{⟨label⟩}{⟨title⟩}
```

If not set, the handler will try \⟨*label*⟩groupname (for compatibility with glossaries) and if that's not defined the label will be used as the title.

If the location field is set then that value will be used as the location list otherwise if loclist is set then the list given by that field will be iterated over using the same method used by the handler for \printnoidxglossary (which is quite primitive, as can be seen in the results for Examples 23, 24 and 27 in Table 10).

It's therefore possible to manually produce a glossary with groups and locations like this:

```
\documentclass{article}

\usepackage{glossaries-extra}

\setglossarystyle{indexgroup}
\renewcommand*{\glstreenamefmt}[1]{#1}

\glsaddstoragekey{group}{}{\glsgroup}

\glsxtrsetgrouptitle{42}{B}
\glsxtrsetgrouptitle{D8}{\O}

\newglossaryentry{books}
{name={books},group={42},description={}}

\newglossaryentry{books.Dubliners}
{name={\emph{Dubliners}},parent={books},
description={}}
\GlsXtrSetField{books.Dubliners}{location}
{1--3}

\newglossaryentry{books.Ulysses}
{name={\emph{Ulysses}},parent={books},
description={}}
\GlsXtrSetField{books.Ulysses}{location}{2}

\newglossaryentry{OlstykkeStenlose}
{name={\O lstykke-Stenl\o se},group={D8},
description={}}
\GlsXtrSetField{OlstykkeStenlose}{location}{8}

\newglossaryentry{Oresund}
{name={\O resund},group={D8},description={}}
\GlsXtrSetField{Oresund}{location}
{9, \emph{see also} \O resund Bridge}

\begin{document}
\printunsrtglossary
\end{document}
```

This produces:

B

books

Dubliners 1–3

Ulysses 2

Ø

Ølstykke-Stenløse 8

Øresund 9, *see also* Øresund Bridge

On the face of it, this method seems contrary to one of LaTeX's biggest advantages in its ability to automate cross-referencing and indexing. However, it's just this method that's used by `bib2gls`, which performs two tasks:

1. fetches entry information from a `.bib` file;
2. performs hierarchical sorting, optionally assigns letter groups, collates location lists and writes the entry definitions to a file that can be input by `\GlsXtrLoadResources`.

The first task is akin to using `bibtex` or `biber`. The second task is similar to that performed by `makeindex` or `xindy`.

The LaTeX code generated by `bib2gls` has the entry definitions written in the order obtained from sorting, with parent entries defined immediately before their child entries. The information required by `bib2gls` is provided in the `.aux` file, but this needs to be enabled by passing the `record` option to glossaries-extra.

An additional build may be required to ensure the locations are up-to-date as the page-breaking may be slightly different on the first LaTeX run due to unknown references being replaced with '??', which can be significantly shorter than the actual text produced when the reference is known.

The command `\glsaddall` can't be used in this mode, but it's possible to instruct `bib2gls` to select all entries. By default it only selects those entries that have been indexed and their dependencies (which includes their ancestors). Since only the required entries have been defined and they have been defined in the correct order, the glossary can be displayed using `\printunsrtglossary`.

▸ **Example 29** (testidx-glossaries and `bib2gls`)
This example uses `bib2gls`, so this needs:

```
\usepackage[bib2gls]{testidx-glossaries}
```

The entries are defined in various `.bib` files provided with testidx. The test document is:

```
\documentclass{article}

\usepackage[a4paper]{geometry}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{amssymb}
\usepackage[bib2gls]{testidx-glossaries}

\tstidxmakegloss

\renewcommand*{\glstreenamefmt}[1]{#1}
\renewcommand*{\glstreegroupheaderfmt}[1]{%
 \textbf{#1}}

\begin{document}
\testidx
\tstidxprintglossaries
\end{document}
```

The document build process is:

```
pdflatex doc
bib2gls --group doc
pdflatex doc
```

The `--group` switch enables the letter group formation, which is off by default. Note that UTF-8 support is needed with this switch as the groups may contain extended characters. The build times shown in Table 14 use the above build sequence for the `bib2gls` examples. However, the first instance (or when new entries are referenced) will need:

```
pdflatex doc
bib2gls --group doc
pdflatex doc
bib2gls --group doc
pdflatex doc
```

to ensure the location lists are correct. The `.log` file will warn about undefined references on the first run, so build processes that allow for conditional actions can perform a check for these warnings. For example, using `arara` v4.0:

```
% arara: pdflatex
% arara: bib2gls if found ("log", "Warning:
Glossary entry")
% arara: pdflatex if found ("log", "Warning:
Glossary entry")
% arara: bib2gls: {group: on}
% arara: pdflatex
```

The file `testidx-glossaries-samples-ascii.bib` contains definitions using commands for extended characters, for example:

```
@index{elite,
  name={{\'e}lite},
  category={word},
  description={group of people regarded
  as the best of a particular society
  or organisation}
}
```

(The initial `\'e` is grouped to allow it to work with the case-changing `\Gls`.) None of the sample `.bib` files provide a `sort` key, but `bib2gls` has a primitive

TEX interpreter that recognises accent commands, so it determines that the sort value for this entry is élite. This means that it can place this word in the E letter group (if appropriate to the collation rule). In the case of \O resund, bib2gls determines that it belongs to the Ø letter group (again, depending on the rule). Since with inputenc Ø is an active character, bib2gls uses numeric identifiers as the group labels (to avoid problems with hyperref). Although the entry definition is written with the original \O used in the .bib file, the letter group title is an extended character taken from the sort value, which is why either UTF-8 support is needed or the --group option should be omitted.

In ASCII mode, \tstidxmakegloss selects the *-ascii.bib file, whereas with UTF-8 support, this command selects UTF-8 versions (*-utf8.bib) and terms such as élite no longer need the interpreter. (Only terms containing \ { } or $ are passed to the interpreter.)

The definition of the test interface command \tstidxmakegloss varies according to the package options. If you add the verbose option, the transcript will list the exact sequence of resource commands. So for this example, the .log file includes:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-mathsym},
 group={Maths},
 sort={letter-case},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

This mimics the prefix setting used in earlier examples. The maths symbols are defined in the file testidx-glossaries-mathsym.bib like this:

```
@symbol{spinderiv,
  name={$\eth$},
  text={\eth},
  category={mathsymbol},
  description={spin-weighted partial
  derivative}
}
```

Entries defined using @symbol or @number fall back to the *label* if the sort field is missing. This means that ð now has a different sort value (spinderiv) from the earlier examples where it was either >eth or eth. This is reflected in Table 4 where the ordering has changed.

The value of the src key identifies the .bib file (where the extension is omitted). This may be a comma-separated list. The group key sets the group field for all the selected entries, which overrides the default method of obtaining the group from the entry's sort value. (This will be ignored if bib2gls is run without the --group switch.) The sort setting letter-case indicates case-sensitive letter order.

The selection value recorded and deps and see instructs bib2gls to select all entries that have been indexed (recorded) in the document (through commands like \gls) and their dependencies (such as parent entries) and their cross-references. This ensures that sub-entries, such as books.Ulysses, have their parent entry listed. The hierarchical sort ensures the sub-entries are defined immediately after their parent entry to keep them together.

The final key ignore-fields tells bib2gls to ignore the description field (to honour the default nodesc package option). The @index entry type allows a missing description, unlike the @entry type (not used in any of the provided .bib files) which requires that field.

The above is the first resource command, which instructs bib2gls to create a file called doc.glstex (where the main document file is called doc.tex) with the required definitions in the appropriate order. A separate file is created for each instance of \GlsXtrLoadResources. This allows different ordering within sub-units of the glossary (or index). The use of the group key assigns the sub-unit to a single group.

The next resource command is quite similar:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-markers},
 group={Markers},
 sort={letter-case},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

This loads the .bib file that contains the definitions of all the markers, again using @symbol. The LATEX code is written to doc-1.glstex.

The third command is:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-numbers},
 sort={integer},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

This loads the .bib file that contains the definitions of all the numbers in the form:

```
@number{10,
  name={10},
  category={number},
  description={ten}
}
```

The sort key has been set to integer to order these entries numerically. This automatically assigns them to the 'Numbers' group so no group option is used here. The LATEX code for this resource set is written to doc-2.glstex.

The final resource command is:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-samples,
      testidx-glossaries-samples-utf8,
      testidx-glossaries-nodiglyphs-utf8},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

The `.bib` files listed in `src` vary according to the testidx-glossaries package options and document encoding. There's no `sort` option in this resource set. The glossaries package loads tracklang [11] (described in a previous issue of *TUGboat* [6]). If a document language is detected, glossaries-extra will use the tracklang interface to write the locale information to the `.aux` file, which `bib2gls` will detect and will use as the default sort. If there is no document language (as in this case), `bib2gls` will fall back on the operating system's locale. In my case, this is `en-GB` so the entries will be sorted according British English. Another user with a different locale may find that the resulting letter groups are different to those shown in Table 1. The optional argument of `\tstidxmakegloss` is appended to this final instance of `\GlsXtrLoadResources` (but not to any of the others), so to replicate this example, you can do

```
\tstidxmakegloss[sort=en-GB]
```

(or just `sort=en`).

The non-native (for English) letters Ø and Ł have been combined into a single group after Z. The rules used by `sort=`⟨*locale*⟩ are in the form ⟨*ignore chars*⟩ < ⟨*char group 1*⟩ < ⟨*char group 2*⟩ .... (You can see the rule in the transcript by running `bib2gls` with the `--debug` switch.) Any characters that don't appear in the rule (such as Ø and Ł) are always placed at the end of the alphabet. `bib2gls` determines the letter group title from the first entry in the group.

The remaining letter groups in this example are sensible for this locale as they are included in the `en` rule. Đ is placed between D and E, and ß is treated as 'ss' (Table 8).

The sort value for each entry is converted to a set of collation keys, where each key is an integer representing a 'letter' as defined by the collation rule. The letter may be more than one character, for example, if the rule includes digraphs or trigraphs. Ignored characters aren't included in the key set. The comparison is performed on this key set rather than on the sort string.

Group titles are determined by taking the first collation key from the set and looking up the corresponding sub-string from the sort value. This sub-string is then converted to lower case and any modifiers are stripped using a normalizer (where possible). If the result is considered equivalent to the original sub-string according to the collator, then the normalized version is considered the group title and the first character is converted to upper case (except for the Dutch 'ij', which is converted to IJ, see Example 33). For example, the first letter of élite is 'é' which is normalised to 'e'. Since the sort rule considers é and e to belong to the same letter group, the group title becomes E. In the case of Øresund, the result of the normalisation 'o' doesn't match the original, so the group title is Ø.

The multiple encap (Table 9) generates a warning from `bib2gls`. It gives precedence to the first non-default of the conflicting set (`tstidxencapi`, in this case). Precedence can be given to a different encap through the `--map-format` switch.

The range interruption has been moved before the start of the explicit range (Table 10) but the explicit range 2–5 (created with the open and close formats) hasn't been merged with the individual locations 1 and 6 on either side of it. The `notestencaps` option doesn't use any of the test encaps, so with

```
\usepackage[bib2gls,notestencaps]
  {testidx-glossaries}
```

the interrupting entry now has the same format as the explicit range. This means that it can be absorbed into the range, but an explicit range doesn't merge with neighbouring locations, so the location list becomes 1, 2–5, 6.

The space and hyphen characters are in the ⟨*ignore chars*⟩ part of the rule. This means that the locale sorting naturally used by Java (in which `bib2gls` is written) is typically letter order. To implement word-ordering, the sort value is split on word boundaries and joined with `|` (which is usually in its own letter group before digits). For example, 'sea lion' becomes `sea|lion|` (there's always a final marker so 'seal' becomes `seal|`). This ensures that `bib2gls` defaults to word ordering, matching `makeindex` and `xindy` (Table 7). Java's word iterator doesn't consider hyphens as word boundaries so 'yo-yo' becomes `yo-yo|`.

▸ **Example 30** (testidx-glossaries, `bib2gls` and letter order)

In this example, the insertion of the break points is disabled:

```
\tstidxmakegloss[sort=en-GB,break-at=none]
```

This results in letter ordering (Table 7). Note that this isn't the same as `sort=letter-case` which simply sorts according to the Unicode values rather than according to a rule.

The 'L' letter group includes the `-l` and `-L` switches (Table 1), but these are in a different order (Table 3) than the previous example. In this case

`-l (makeindex)` appears at the start of the group whereas in the previous case it came between `-L icelandic (xindy)` and `-L polish (xindy)`.

▸ **Example 31** (testidx-glossaries, `bib2gls` and Icelandic)
For comparison with Example 15, this example sorts according to the Icelandic alphabet:

```
\tstidxmakegloss[sort=is]
```

This correctly identifies all the Icelandic letter groups as shown in Table 1. (There's no Ó or Ý letter group as there are no terms starting with those letters.) The non-native letters C, Œ, Q, W, Z and Ł have also been assigned their own letter groups. The ordering of 'resume' and 'résumé' (Table 6) is different from the previous example since É comes after E in the Icelandic alphabet (and are considered separate letters). They are no longer collation-level homographs. The non-native ß is treated as 'ss' (Table 8).

▸ **Example 32** (testidx-glossaries, `bib2gls` and Hungarian)
For comparison with Example 16, this example sorts according to the Hungarian alphabet:

```
\tstidxmakegloss[sort=hu]
```

In addition to the basic Latin letters A–Z, the Hungarian alphabet also has Á, Cs, Dz, Dzs, É, Gy, Í, Ly, Ny, Ó Ö, Ő, Sz, Ty, Ú, Ü, Ű and Zs. The sample entries don't include any terms starting with Cs, Gy, Ny, Ő, Sz, Ty, Ü, Ű or Zs. Of the other letters, only Ly and Ö have correctly formed letter groups (Table 1). The non-native letters Đ and Œ have formed separate groups, ß has been treated as 'sz' rather than 'ss' (Table 8), and Ø and Ł are collected at the end of the alphabet as they aren't in the rule-set.

This has more success than `xindy` at forming a digraph letter group (Ly) but has missed the Dz digraph and Dzs trigraph.

Since I have Java 8 installed, the above examples are using the locale rules from the CLDR (Common Locale Data Repository). The results may differ with Java 7 which can only use the locale information provided with the JRE (Java Runtime Environment). The locale identifier can include a variant as well as a region, for example, `sort=de-CH-1996` indicates Swiss German new orthography.

▸ **Example 33** (testidx-glossaries, `bib2gls` with custom rules)
This example requires some customisation, so I can't use the convenient `\tstidxmakegloss`. I need to let testidx-glossaries know this with the `manual` option to prevent an error occurring:

```
\usepackage[bib2gls,manual]{testidx-glossaries}
```

I also need to explicitly use

```
\printunsrtglossary
```

So far, the maths group (where it has been formed) only contains symbols such as α. There are some other maths terms that have a natural alphabetic ordering (such as $f(\vec{x})$ and $\boldsymbol{E}$) which have been placed in the letter groups. This example gathers them all together into a single group. As mentioned earlier, terms like α have the `category` set to `mathsymbol`. The other mathematical terms are in `testidx-glossaries-samples.bib` and have the `category` set to `math`. It's possible to apply a filter so that only these terms are selected:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-mathsym,
      testidx-glossaries-samples},
 group={Maths},
 sort={letter-case},
 sort-field={name},
 match-op={or},
 match={{category=mathsymbol},{category=math}},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

I've set the sort field to `name`, which means that `bib2gls` will try to interpret the TEX code. It recognises standard maths commands like `\alpha` and can also detect a limited number of packages, such as amssymb. This means that the sort code for ð becomes the Unicode character F0 (eth).

The markers use the same code shown in Example 29. After that is the number group, which is much the same, but for illustrative purposes, I've inverted the number ordering:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-numbers},
 sort={integer-reverse},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

Next I want to create a group for the switches. The switches also occur as sub-entries (under the name of the application), so I need to select those switches that don't have a parent:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-samples},
 group={Switches},
 sort={letter-nocase},
 match-op={and},
 match={{category=applicationoption},{parent={}}},
 selection={recorded and deps and see},
 ignore-fields={description}]
```

I've used the case-insensitive letter sort which first converts the sort key to lower case and then behaves like `letter-case`.

The remaining entries are the alphabetic terms. The terms that have been previously selected will be ignored (with a warning) as duplicates. I've used a custom sort rule here:

```
\GlsXtrLoadResources[
 src={testidx-glossaries-samples,
      testidx-glossaries-samples-utf8,
      testidx-glossaries-nodiglyphs-utf8},
 selection={recorded and deps and see},
 ignore-fields={description},
 max-loc-diff=3,
 sort=custom,
 sort-rule={' ' < ',' < '(' < ')' < '/' < '|' < '-'
  < a,A & AE,\string\uE6,\string\uC6 % \ae
   & \string\uE1,\string\uC1 % \'a
   & \string\uE4,\string\uC4 % \"a
   & \string\uE5,\string\uC5 % \aa
  < b,B
  < c,C & \string\u107,\string\u106 % \'c
  < d,D < dd,Dd,DD
  < dz,Dz,DZ < dzs,Dzs,DZS
  < \string\uF0,\string\uD0 % \dh
  < e,E & \string\uC9,\string\uE9
  < f,F < ff,Ff,FF < g,G < h,H
  < i,I & \string\uED,\string\uCD % \'i
  < ij,IJ < j,J < k,K < l,L < ll,Ll,LL
  < ly,Ly,LY < m,M < n,N < ng,Ng,NG
  < o,O & OE,\string\u153,\string\u152 % \oe
    & \string\uF6,\string\uD6 % \"O
  < p,P < q,Q < r,R
  < s,S & SS,\string\uDF
    & \string\u15B,\string\u15A % \'s
  < t,T
  < th,\string\uFE,Th,TH,\string\uDE % \th
  < u,U & \string\uFA,\string\uDA % \'U
  < v,V < w,W < x,X < y,Y
  < z,Z & \string\u17C,\string\u17B % \.Z
  < \string\uF8,\string\uD8 % \o
  < \string\u142,\string\u141 % \l
 }
]
```

The `sort=custom` option requires the `sort-rule` key to be also set. Extended characters can be identified with `\u`⟨*hex*⟩ but `\string` is needed to prevent expansion when the information is written to the `.aux` file. With XeLaTeX or LuaLaTeX the characters can be written directly.

This rule has only a limited number of punctuation characters for brevity. Extra characters should be added to the rule if required. This is the only example that successfully creates the Dzs trigraph letter group (Table 1). There are also letter groups for the Welsh Dd, Ff, Ll and Ng digraphs, the Dutch IJ digraph, and the Hungarian Dz and Ly digraphs (although the word beginning with 'ly' is actually Polish). There's also a group for both þ and the Th digraph. The eszett ß has been treated as 'ss' (Table 8).

I've listed the hyphen immediately before A (and after the break point marker), which affects the ordering of the compound words (Table 7). This also means that 'recover' and 're-cover' are no longer collation-level homographs (Table 6) since the hyphen is no longer ignored.

The additional `seealso` key provided by v1.16 of glossaries-extra allows `bib2gls` to treat the `see` and `seealso` cross-references differently. (An entry may have one or the other of those fields, but not both with `bib2gls`.) The `seealso` field can be positioned at the start of the location list using the resource option `seealso=before` or omitted entirely using `seealso=omit`. The default setting is `seealso=after`, which puts it at the end of the list. The separator between the list and the cross-reference is given by `\bibglsseealsosep`, which can be redefined after the resources are loaded. In this example, I've done:

```
\renewcommand*{\bibglsseealsosep}{ }
\renewcommand*{\glsxtruseseealsoformat}[1]{%
 (\glsseeformat[\seealsoname]{#1}{})}
```

This puts the 'see also' cross-references in parentheses, but doesn't affect the 'see' cross-references. For example, 'range separator' is defined with the `see` field, and the result is 'range separator *see* location list', but 'padding' is defined with the `seealso` field, so the result is 'padding 2 (*see also* filler)'.

Implicit ranges are formed from consecutive locations. This can lead to some ragged location lists, such as 1, 2, 4, 5, 7. A tidier approach is to show this as 1–7 passim, where 'passim' indicates the references are scattered here and there throughout the range. The `max-loc-diff` option indicates the maximum difference between two locations to consider them consecutive. The default value is 1, which means that 2 and 3 are consecutive but 2 and 4 aren't. I've set the value to 3 in this example, which means that the location list 2, 5, 6 can be tidied into 2–6 passim. The ragged list for 'paragraph' (Table 9) can't be tidied as there are different encaps. The 'passim' suffix can be altered or removed as required.

▸ **Example 34** (testidx-glossaries, `bib2gls` and non-standard page numbering)
This example uses the same custom `\tally` command from Example 22 for the page numbering. The only modification to Example 33 is the addition of:

```
\usepackage[notext]{stix}
```

and the definition of `\tally` and `\thepage` from Example 22.

`bib2gls` will allow any location format. If it can deduce an associated numeric value, it will try to determine if a range can be formed, otherwise the location will be considered an isolated value that can't be concatenated. (With glossaries-extra, it's possible to override the normal location value when

using `thevalue` with `\gls...`, for example, `\glsadd[thevalue={Suppl.\ info.}]{`⟨*label*⟩`}`.) One of the patterns `bib2gls` checks for is `\`⟨*csname*⟩`{`⟨*num*⟩`}`, which it interprets as having the numeric value ⟨*num*⟩. The regular expression for ⟨*num*⟩ can detect roman numerals (I, II, ... or i, ii, ...) or numeric values or single alphabetical characters.

The alphabetic test uses `\p{javaUpperCase}` for the upper case version or `\p{javaLowerCase}` for the lower case version which not only matches A, B, etc., or a, b, etc., but also matches alphabetic characters in other scripts, such as А, Б, etc. The numeric value representing the location is obtained from the Unicode value. For example, Latin A has the value 65 whereas Cyrillic A has the value 1040.

The numeric test uses `\p{javaDigit}` to match a digit, which means it not only matches the digits 0, 1, 2, etc., but also digits from other scripts, such as the Devanagari numbering system ०, १, २, etc.

The results from this example are much the same as the previous example except for the page number representation (Tables 9, 10, 11 and 12).

5 Extending the dummy text

New blocks can be added using `\tstidxnewblock`. For example:

```
\tstidxnewblock{The \tstidxword{cat} sat
on the \tstidxword{mat}. The
\tstidxphrase{man in the moon} fell off
the \tstidxphrase{four-poster bed}.}
```

The starred version can be used to capture the block number in a control sequence:

```
\tstidxnewblock*{\moonblock}{The
\tstidxword{cat} sat on the \tstidxword{mat}.
The \tstidxphrase{man in the moon} fell off
the \tstidxphrase{four-poster bed}.}
```

You can then display just this block with

```
\testidx[\moonblock]
```

There are other commands as well, including commands for UTF-8 terms. For example:

```
\tstindexutfword{ch\^ateau}[chateau]{château}
```

The first argument is the ASCII version and the final argument is the UTF-8 version. The optional argument is the label, which is only used by testidx-glossaries. If you want this support for the glossaries package, you'll need to define the terms as well:

```
\tstidxnewword{cat}{feline animal}
\tstidxnewword{mat}{piece of material
placed on the floor}
\tstidxnewphrase{man in the moon}{pareidolic
image seen in the moon}
\tstidxnewphrase{four-poster bed}{type of bed}
```

The UTF-8 example needs to be defined as follows:

```
\tstidxnewutfword{chateau}{ch\^ateau}{château}
{castle}
```

where the first argument is the label.

To integrate this with `\tstidxmakegloss`, just add the definition file name to the comma-separated list given by `\tstidxtexfiles`. For example (using etoolbox), if the terms are defined in the file `my-samples.tex`:

```
\appto{\tstidxtexfiles}{,my-samples}
```

With `bib2gls`, the definitions will need to go in a `.bib` file. For example:

```
@index{cat,
  category={word},
  description={feline animal}
}
@index{fourposterbed,
  category={phrase},
  name={four-poster bed},
  description={type of bed}
}
```

(Note that the hyphen and space are stripped from the name to create the label. The `name` field may be omitted if it's identical to the label.) The UTF-8 support is dealt with by having two separate `.bib` files. One contains the ASCII version:

```
@index{chateau,
  category={word},
  name={ch\^ateau},
  description={castle}
}
```

and the other contains the UTF-8 version:

```
@index{chateau,
  category={word},
  name={château},
  description={castle}
}
```

These can also be integrated into `\tstidxmakegloss` as follows. The `.bib` file that doesn't require UTF-8 support (the one containing 'cat' in the above) needs to be added to `\tstidxbasebibfiles` (a comma-separated list). For example, if that file is called `my-samples.bib` then:

```
\appto{\tstidxbasebibfiles}{,my-samples}
```

The UTF-8 file (the one containing `château`) needs to be added to `\tstidxutfbibfiles`. For example, if the file is called `my-samples-utf8.bib`:

```
\appto{\tstidxutfbibfiles}{,my-samples-utf8}
```

and the corresponding ASCII file needs to be added to `\tstidxasciibibfiles`. For example, if the file is called `my-samples-ascii.bib`:

```
\appto{\tstidxasciibibfiles}{,my-samples-ascii}
```

Table 1: Letter groups

Example	Group ordering
1, 5, 18, 19	Symbols Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. ð) E (inc. é) F G H I (inc. Í) J K L (inc. Ł) M N O (inc. œ, Ø, Ö) P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż)
2	Symbols Numbers A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3	Symbols Numbers A (inc. æ, α, Á, Ä, Å) B (inc. β) C (inc. Ć) D (inc. ð) E (inc. é, ð) F G (inc. γ) H I (inc. Í) J K L (inc. Ł) M N O (inc. œ, Ø, Ö) P (inc. ∂) Q R S (inc. Ś, $\sum$) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż)
4	Symbols Numbers A (inc. α) B (inc. β) C D E (inc. ð) F G (γ) H I J K L M N O P (inc. ∂) Q R S (inc. $\sum$) T U V W X Y Z
6	Symbols Numbers A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Other (Á, Ä, Å, Í, Ö, Ø, Ú, Þ, æ, é, ð, þ) Other (Ć) Other (Ł, œ, Ś, Ż)
7–9	Symbols A (inc. æ) B C D (inc. ð) E (inc. é) F G (inc. `-g`) H I J K L (inc. `-l`, `-L`) M (inc. `-M`) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z Þ
10	Symbols A (inc. æ, α) B (inc. β) C D (inc. ð) E (inc. é, ð) F G (inc. `-g`, γ) H I J K L (inc. `-L`, `-l`) M (inc. `-M`) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P (inc. ∂) Q R S (inc. $\sum$) T U V W X Y Z Þ
11	Symbols Numbers A (inc. æ) B C D (inc. ð) E (inc. é) F G (inc. `-g`) H I J K L (inc. `-l`, `-L`) M (inc. `-M`) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z Þ
12	Symbols Numbers A (inc. æ, α) B (inc. β) C D (inc. ð) E (inc. é, ð) F G (inc. `-g`, γ) H I J K L (inc. `-l`, `-L`) M (inc. `-M`) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P (inc. ∂) Q R S (inc. $\sum$) T U V W X Y Z Þ
13	Symbols Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. ð) E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-l`, `-L`, Ł) M (inc. `-M`) N O (inc. œ, Ø, Ö) P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż)
14, 21, 22	Symbols Markers Maths Numbers A (inc. æ) B C D (inc. ð) E (inc. é) F G (inc. `-g`) H I J K L (inc. `-l`, `-L`) M (inc. `-M`) N O (inc. Á, Ä, Å, Í, œ, Ø, Ú, Ö) P Q R S T U V W X Y Z Þ
15 (Icelandic)	Symbols Markers Maths Numbers A (inc. Á) B C D ð E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-l`, `-L`) M (inc. `-M`) N O P Q R S T U (inc. Ú) V W X Y Z Þ Æ (inc. Ä, œ) Ö (inc. Ø) Å
31 (Icelandic)	Maths Markers Numbers A (inc. Ä, Å) Á B C (inc. Ć) D Ð E É F G (inc. `-g`) H I Í J K L (inc. `-L`, `-l`) M (inc. `-M`) N O Œ P Q R S (inc. Ś) T U Ú V W X Y Z (inc. Ż) Þ Æ Ö (inc. Ø) Ł
16 (Hungarian)	Symbols Markers Maths Numbers A (inc. Á) B C D (inc. dz, dzs) E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-l`, `-L`, ly) M (inc. `-M`) N O (inc. Ä, Å, Ø, Þ, æ, ð, þ) Ö P Q R S T U (inc. Ú) V W X Y Z
32 (Hungarian)	Maths Markers Numbers A (inc. æ Á Ä Å) B C (inc. Ć) D (inc. dz, dzs) Ð E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-L`, `-l`) Ly M (inc. `-M`) N O Ö Œ P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż) Ø (inc. Ł)
17	Markers Maths Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć), D (inc. dd and dzs) Dz E (inc. é) F (inc. ff) G (inc. `-g`) H I (inc. Í) IJ J K L (inc. `-l`, `-L`, Ł, ly) IL M (inc. `-M`) N (inc. Ng) O (inc. œ, Þ, ð, Ø, Ö, þ) P Q R S (inc. Ś) T U (inc. Ú) V W X Y Z (inc. Ż)
20	Markers Maths Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. ð) E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-l`, `-L`, Ł) M (inc. `-M`) N O (inc. œ, Ø, Ö) P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż)
23, 24	Symbols A (inc. æ, Á, Ä, Å) B C (inc. Ć) D (inc. ð) E (inc. é) F G H I (inc. Í) J K L (inc. Ł) M N O (inc. Ø, Ö) P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż)
29, 30	Maths Markers Numbers A (inc. æ, Á, Ä, Å) B C (inc. Ć) D Ð E (inc. é) F G (inc. `-g`) H I (inc. Í) J K L (inc. `-L`, `-l`) M (inc. `-M`) N O (inc. œ, Ö) P Q R S (inc. Ś) T (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż) Ø (inc. Ł)
33, 34	Maths Markers Numbers Switches A (inc. æ, Á, Ä, Å) B C (inc. Ć) D Dd Dz Dzs Ð E (inc. é) F Ff G H I (inc. Í) IJ J K L Ll Ly M N Ng O (inc. œ, Ö) P Q R S (inc. Ś) T Th (inc. þ, Þ) U (inc. Ú) V W X Y Z (inc. Ż) Ø Ł
25–28	*no groups or all entries in one group*

Table 2: Symbols

Example	Symbol group contents
1, 5, 18, 19	switches markers maths
2	switches markers maths non-ASCII (Ä, Ö, Á, Ć, Í, Ś, Ú, é, Ż, Ł, Ø, Þ, æ, ð, œ, Å, þ)
3	switches
4	switches non-ASCII (Ä, Ö, Á, Ć, Í, Ś, Ú, é, Ż, Ł, Ø, Þ, æ, ð, œ, Å, þ)
6	switches markers maths
7–9	numbers markers maths UTF-8 (Ć, Ł, Ś, Ż)
10	numbers UTF-8 (Ć, Ł, Ś, Ż)
11	markers maths UTF-8 (Ć, Ł, Ś, Ż)
12, 14, 15, 21, 22	UTF-8 (Ć, Ł, Ś, Ż)
13	markers maths
16	UTF-8 (Ć, Ł, œ, Ś, Ż)
23, 24	switches markers maths numbers
17, 20, 25–34	*group missing*

Table 3: Switches

Example	Switches ordering
1–6, 18, 19, 23, 24	`-L -M -g -l`
7–9, 11–17, 20–22, 30, 33, 34	`-g -l -L -M`
10, 29, 31, 32	`-g -L -l -M`
25–28	*order of definition*

Table 4: Maths

Example	Maths ordering
1, 2, 5–11, 13–24	α (`>alpha`), β (`>beta`), ð (`>eth`), γ (`>gamma`), ∂ (`>partial`), $\sum$ (`>sum`).
3, 4, 12	α (`alpha`), β (`beta`), ð (`eth`), γ (`gamma`), ∂ (`partial`), $\sum$ (`sum`).
29–32	α (`alpha`), β (`beta`), γ (`gamma`), ∂ (`partial`), ð (`spinderiv`), $\sum$ (`sum`).
33, 34	E (45), $f(\vec{x})$ (66 28 78 20D7 29), n (6E), ð (F0), ∂ (2202), $\sum$ (2211), α (1D6FC), β (1D6FD), γ (1D6FE).
25–28	*order of definition*

Table 5: Numbers

Example	Number ordering
1–6, 11–24, 29–32	2, 10, 16, 42, 100
7–10	10, 100, 16, 2, 42
33, 34	100, 42, 16, 10, 2
25–28	*order of definition*

Table 6: Collation-level homographs

Example	Ordering
1–6	`imakeidx` package, **index**, `\index`, indexing application
7–17	`imakeidx` package, `\index`, indexing application (*'index' omitted*)
18, 19, 23, 24, 29–34	illustration, `\index`, **index**, indexing application
20–22	illustration, `\index`, indexing application (*'index' omitted*)
1–6, 18, 19, 23, 33, 34	range separator, **re-cover**, **recover**, reference
7–17, 24, 20–22, 29–32	range separator, **recover**, **re-cover**, reference
1, 3, 5, 7–12, 14–17, 21, 22, 31	repetition, **resume**, **résumé**, rhinoceros
18, 19, 23, 24, 29, 30, 32–34	repetition, **résumé**, **resume**, rhinoceros
2, 4	(*start of group*) **résumé**, Rødovre, raft, . . . , repetition, **resume**, rhinoceros
6	repetition, **resume**, rhinoceros, . . . , roundabout, **résumé**, Rødovre
13, 20	repetition, **résumé**, rhinoceros (*'resume' omitted*)
25–28	*order of definition*

Table 7: Compound words

Example	Ordering
1–4, 6, 7, 9–22, 29, 31–34	sea, sea lion, seaborne, seal, sealant gun
5, 8, 24, 30	sea, seaborne, seal, sealant gun, sea lion
23	sea, sealant gun, sea lion, seaborne, seal
1–4, 6, 7, 9–22, 29, 31–34	vice admiral, vice chancellor, vice versa, vice-president, viceregal, viceroy
5	vice-president, vice admiral, vice chancellor, viceregal, viceroy, vice versa
8, 24, 30	vice admiral, vice chancellor, vice-president, viceregal, viceroy, vice versa
23	vice chancellor, viceregal, vice versa, vice admiral, vice-president, viceroy
1–6, 18, 19, 33, 34	yawn, **yo-yo**, yoghurt
7–17, 20–22, 24, 29–32	youthful, **yo-yo**, yuck
23	yoghurt, **yo-yo**, youthful
25–28	*order of definition*

Table 8: Eszett (‘Aßlar’)

Example	Ordering
1, 3, 5, 13, 15, 17–20, 23, 24, 29–31, 33, 34	assailed, **Aßlar**, astounded
2, 4	(*start of group*) **Aßlar**, aardvark
6	attributes, **Aßlar** (*end of group*)
7–12, 14, 16, 21, 22	anonymous reviewer, **Aßlar**, applications
32	astounded, **Aßlar**, attaché case
25–28	*order of definition*

Table 9: Multiple encap (‘paragraph’)

Example	Location List
1–6	2, 2, 2, 2, 3, 5
7–10	2, 2, 3, 5
11–16	2, 3, 5
17	2, 3, 4, 6
18	2, 3, 3, 3, 3, 4, 6
19, 25	2, 3, 4, 6
20, 21, 26, 29–33	2, 3, 4, 6
22, 34	⚁, ⚂, ⚃, ⚄
23, 24, 27	2, 2, 2, 2, 3, 6
28	*locations missing*

Table 10: Explicit range interruption (‘range’)

Example	Location List
1–6	2, 1–4, 6
7–16	1–4, 2, 6
17	1, 2, 6
18, 19, 25	3, 1–6
20, 21, 26	1–6, 3
22	⚀–⚄, ⚂
23, 24, 27	1, 2, 2,5, 6
29–33	1, 3, 2–5, 6
34	⚀, ⚂, ⚁–⚅, ⚄
28	*locations missing*

Table 11: Cross-reference interruption (‘lyuk’)

Example	Location List
1–6,	1, *see also* digraph, 3
7–17	1, 3, *see also* digraph
18–21, 25, 26, 29–32	2, 3, *see also* digraph
22	⚁, ⚂, *see also* digraph
23, 24, 27	*see also* digraph, 1, 3
33	2, 3 (*see also* digraph)
34	⚁, ⚂ (*see also* digraph)
28	*locations missing*

Table 12: Ragged page list (‘block’)

Example	Location List
1–16	1, 2, 4–6
17–21, 23–27, 29–32	2, 5, 6
22	⚁, ⚅, ⚄
33	2–6 passim
34	⚁–⚄ passim
28	*locations missing*

Table 13: Build time (testidx)

Example	Elapsed real time	External tool
1	0:00.73	makeindex
2	0:00.64	makeindex
3	0:00.64	makeindex
4	0:00.69	makeindex
5	0:00.56	makeindex
6	0:00.61	makeindex
7	0:01.17	xindy
8	0:01.13	xindy
9	0:01.07	xindy
10	0:01.13	xindy
11	0:01.12	xindy
12	0:01.32	xindy
13	0:01.38	xindy
14	0:01.19	xindy
15	0:01.13	xindy
16	0:01.17	xindy
17	0:02.43	xindy

Table 14: Build time (testidx-glossaries)

Example	Elapsed real time	External tool
18	0:02.45	makeglossaries-lite
	0:02.08	makeindex (explicit)
19	0:02.42	makeglossaries (makeindex)
20	0:03.19	makeglossaries (xindy)
21	0:03.18	makeglossaries (xindy)
22	0:03.29	makeglossaries (xindy)
23	3:31.69	*none*
24	3:34.38	*none*
25	0:02.24	makeglossaries (makeindex)
26	0:03.18	makeglossaries (xindy)
27	0:03.79	*none*
28	0:01.57	*none*
29	0:05.33	bib2gls
30	0:05.08	bib2gls
31	0:05.03	bib2gls
32	0:05.06	bib2gls
33	0:06.04	bib2gls
34	0:05.50	bib2gls

References

[1] STI Pub Companies. The stix package, 2015. ctan.org/pkg/stix.

[2] Alan Jeffrey and Frank Mittelbach. The inputenc package, 2015. ctan.org/pkg/inputenc.

[3] Philipp Lehman and Joseph Wright. The etoolbox package, 2015. ctan.org/pkg/etoolbox.

[4] Frank Mittelbach, Robin Fairbairns, and Werner Lemberg. The fontenc package, 2016. ctan.org/pkg/fontenc.

[5] American Mathematical Society. The amssymb package, 2013. ctan.org/pkg/amsfonts.

[6] Nicola Talbot. Localisation of TEX documents: tracklang. *TUGboat*, 37(3):337–351, 2016. tug.org/TUGboat/tb37-3/tb117talbot.pdf.

[7] Nicola Talbot. bib2gls: A command line Java application to convert .bib files to glossaries-extra.sty resource files, 2017.

[8] Nicola Talbot. The glossaries-extra package, 2017. ctan.org/pkg/glossaries-extra.

[9] Nicola Talbot. The glossaries package, 2017. ctan.org/pkg/glossaries.

[10] Nicola Talbot. The testidx package, 2017. ctan.org/pkg/testidx.

[11] Nicola Talbot. The tracklang package, 2017. ctan.org/pkg/tracklang.

◇ Nicola L. C. Talbot
School of Computing Sciences
University of East Anglia
Norwich Research Park
Norwich
NR4 7TJ
United Kingdom
N.Talbot (at) uea dot ac dot uk
http://www.dickimaw-books.com/