testidx.sty v1.1: dummy text for testing indexes

Nicola L.C. Talbot

http://www.dickimaw-books.com/

2017-08-11

Contents

1 Introduction

The testidx package is for testing indexes (`\index`, theindex and indexing applications, such as `makeindex` and `xindy`). As with packages like lipsum and blindtext, this package provides dummy text, but it's interspersed with `\index` commands. The filler text is mostly English not lorum ipsum, as this makes it slightly easier to check the words in the index against the words in the document. (For those who don't understand English, it's at least no worse than lorum ipsum.) There are some terms (words, phrases or proper nouns) that include extended Latin characters or digraphs to allow for testing with a variety of Latin alphabets. The dummy text is designed to cause problems that can occur in real documents to help test your chosen indexing application or index style. The main issues this package tries to replicate are:

- Multiple encaps. For example, the word "paragraph" is indexed within the same block using no encap and each of the three test encap values. This causes the `makeindex` warning "Conflicting entries: multiple encaps for the same page under same key."

- An explicit range formation conflicting with a mid-range encap. The word “range” has an explicit range formation (starting in block 4 and ending in block 9), but “range” is also indexed in block 5 with one of the test encap values. This causes the `makeindex` warning “Inconsistent page encapsulator ... within range.”

- Page breaking mishaps. This is largely dependent on the font size and page geometry, but the dummy text contains some long paragraphs and has enough entries to result in at least some awkward page breaks. These may include a page or column break between an index group heading and the first entry in that group or between an index item and the first sub-item following it. Also check for indexing that occurs in paragraphs that span page breaks to ensure the location number is correct.

- Untidy page lists. This again depends on the font size and page geometry, but some entries are sporadically indexed throughout the dummy text, which can lead to a long list that can't be formed into a neat range.

- Mid-list cross-referencing. The word “lyuk” is indexed and then cross-referenced in block 3, and indexed again in block 7. This can result in the rather odd occurrence of a cross-reference appearing in the middle of the location list for that entry, depending on the indexing method.

- Collation-level homographs. (Same spelling except for accents.) The words “resume” and “résumé” are both indexed. These should be treated as separate entries in the index, even if the comparator considers them identical. Different indexing methods may produce different ordering or may even merge the two words, so check they are both present.

- Compound entries. The index contains a mixture of single words, compound words, names, titles and phrases. The ordering may vary depending on the sorting method. For example, check the ordering of “sea”, “sea lion”, “seaborne” and “seal”, and also the words starting with “vice”, such as “vice admiral”, “viceroy” and “vice-president”.

- Long entries can cause awkward line breaks and justification in a multicolumn index with narrow columns.

- Interference caused by whatsits. Block 8 has a whatsit caused by the indexing that interferes with limits of a summation in an equation.

- Symbols and numbers that don't have a natural word order. The numbers may or may not be ordered numerically, depending on the indexing method.

In addition, words containing extended Latin characters, digraphs and a trigraph are indexed to help test various Latin alphabets, such as Swedish, Icelandic, Welsh, Dutch, Polish and Hungarian. These may or may not be recognised by indexing applications.

Version 1.1 now comes with a supplementary package testidx-glossaries which provides a similar way of testing the glossaries or glossaries-extra package.

Example document:

```
\documentclass{article}

\usepackage{makeidx}
\usepackage{testidx}

\makeindex

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

If the document is called, say, `myDoc.tex`, then the PDF can be built using:

```
pdflatex myDoc
makeindex myDoc.idx
pdflatex myDoc
```

There will be warnings about multiple encaps. This is intentional to test how the indexing applications deal with this problem.

If you want to use `xindy`, you'll need to define the attributes (encaps) used in the dummy text. For example:

```
\documentclass{article}

\usepackage{filecontents}
\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{makeidx}
\usepackage{testidx}

\begin{filecontents*}{\jobname.xdy}
; 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")

(markup-locref-list :sep ", ")
(markup-range :sep "--")
\end{filecontents*}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

If this document is called, say, `myDoc.tex` then the build process is:

```
pdflatex myDoc
xindy -L english -C utf8 -M myDoc.xdy -M texindy -t myDoc.ilg myDoc.idx
pdflatex myDoc
```

You can substitute `english` for another language (for example, `swedish` or `danish`) to test how the extended Latin characters are sorted for a particular language.

XeLaTeX can be used instead:

```
\documentclass{article}

\usepackage{filecontents}
\usepackage{fontspec}
\usepackage{makeidx}
\usepackage{testidx}

\begin{filecontents*}{\jobname.xdy}
; 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")

(markup-locref-list :sep ",")
(markup-range :sep "--")
\end{filecontents*}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

The build process is now:

```
xelatex myDoc
xindy -L english -C utf8 -M myDoc.xdy -M texindy -t myDoc.ilg myDoc.idx
xelatex myDoc
```

(Similarly for LuaLaTeX.)

If you want to use `makeindex`'s `-g` option (German) you can use the package option german or ngerman, which will change the `makeindex` quote character to + but remember you need to add this to a style file. For example:

```
\documentclass{article}

\usepackage{filecontents}
\usepackage{makeidx}
\usepackage{ngerman}
\usepackage[german]{testidx}

\begin{filecontents*}{\jobname.ist}
quote '+'
\end{filecontents*}

\makeindex

\begin{document}
\testidx
```

```
\printindex
\end{document}
```

This document can be built using:

```
pdflatex myDoc
makeindex -g -s myDoc.sty myDoc.idx
pdflatex myDoc
```

(Note the different position of the “Numbers” group in the index.)

Alternatively:

```
\documentclass[ngerman]{article}

\usepackage{filecontents}
\usepackage{makeidx}
\usepackage{babel}
\usepackage{testidx}

\begin{filecontents*}{\jobname.ist}
quote '+'
\end{filecontents*}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

The testidx-glossaries package automatically loads testidx and will also load either glossaries or glossaries-extra. For example:

```
\documentclass{report}

\usepackage[T1]{fontenc}
\usepackage{testidx-glossaries}

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

\tstidxmakegloss

\begin{document}

\testidx

\tstidxprintglossaries

\end{document}
```

This automatically sets the mcolsindexgroup glossary style to mimic the style commonly used with indexes. This document can be built using:

```
pdflatex myDoc
makeglossaries myDoc
pdflatex myDoc
```

Note that the mcolsindexgroup style sets the name field in \glstreenamefmt, which defaults to bold. This has been redefined in the above example to simply do its argument.

2 Package Options

2.1 testidx options

The following package options are provided:

german or ngerman This redefines the indexing "quote" character to use + instead of the double-quote character. Remember to add this to your style file and call makeindex with the -g (German) switch. (See example above in the previous section.) This option may also be implemented using

\testidxGermanOn

```
\testidxGermanOn
```

nogerman Counteract the effect of the previous option. This option may also be implemented using

\testidxGermanOff

```
\testidxGermanOff
```

stripaccents Strips accent commands from the sort key when using the ASCII option (see Section 5). This option may also be implemented using

\testidxStripAccents

```
\testidxStripAccents
```

Note that the german or ngerman package option won't strip the umlaut accent when used with this option.

nostripaccents Doesn't strip accent commands from the sort key when using the ASCII option (see Section 5). This option may also be implemented using

\testidxNoStripAccents

```
\testidxNoStripAccents
```

sanitize Sanitize the terms before indexing them when using the UTF-8 option to prevent the UTF-8 characters from being expanded to inputenc's internal macros such as `\IeC`. This option is the default unless XeLaTeX or LuaLaTeX are in use. This option may also be implemented using

`\testidxSanitizeOn`

```
\testidxSanitizeOn
```

nosanitize Don't sanitize the terms before indexing them when using the UTF-8 option. This option may also be implemented using

`\testidxSanitizeOff`

```
\testidxSanitizeOff
```

showmarks (Default.) Show the location of the `\index` commands in the dummy text with markers. This option may also be implemented using

`\testidxshowmarkstrue`

```
\testidxshowmarkstrue
```

hidemarks or noshowmarks Hide the markers. This option may also be implemented using

`\testidxshowmarksfalse`

```
\testidxshowmarksfalse
```

verbose Show the actual indexing commands within the dummy text. This will most likely cause a high number of overfull lines. This option may also be implemented using

`\testidxverbosetrue`

```
\testidxverbosetrue
```

noverbose (Default.) Cancel the verbose option. This option may also be implemented using

\testidxverbosefalse

```
\testidxverbosefalse
```

notestencaps Suppress the testing of the encaps. Note that this only affects the commands used within `\testidx`, which have an optional argument to specify the encap. Some of these commands have the default value of the optional argument set to one of the test encaps. This option changes the command definition so that the optional argument is blank. Therefore this setting can only be used as a package option. However, this doesn't prevent you from explicitly testing an encap either directly using `\index` (e.g. `\index{word|emph}`) or implicitly using one of the helper commands described in the documented code (e.g. `\tstidxsty[emph]{testidx}`).

testencaps (Default.) Cancels the notestencaps option. This option ensures that `\testidx` uses the three test encaps.

prefix (Default.) Inserts a prefix in the sort value for certain (symbol) entries to keep them together in the index. These entries represent markers (prefixed with `\tstidxindexmarkerprefix`) and maths symbols (prefixed with `\tstidxmathsymprefix`).

noprefix Doesn't insert a prefix for the markers and maths symbol entries. This option doesn't alter the entries starting with a hyphen (such as `-1`) which always have that prefix since it's part of the display name.

diglyphs Words with "ll", "ij" and "dz" digraphs will have the two characters forming the digraph replaced with a single UTF-8 glyph. This option only works if UTF-8 is supported *and the document font recognises the glyphs*. (The trigraph "dzw" and other digraphs, such as "th" aren't affected by this option.)

nodiglyphs (Default.) Don't use single glyphs for the "ll", "ij" and "dz" digraphs. (This option doesn't affect other glyphs, such as æ or þ, that are more commonly used in some languages.)

2.2 testidx-glossaries options

Most of the package options provided by testidx can also be used with testidx-glossaries. The verbose option has a slightly different effect. With testidx, that option shows the indexing command within the text. However, the glossaries package requires entries to first be defined and doesn't use `\index` but uses its own internal custom commands that depend on the indexing method, so for testidx-glossaries, the verbose option instead writes information in the transcript file (`.log`) when the dummy entries are defined. For example:

```
Package testidx-glossaries Info: new term label={packages},
(testidx-glossaries)             name={packages},
(testidx-glossaries)             text={packages},
(testidx-glossaries)             parent={},
```

```
(testidx-glossaries)             see={}
(testidx-glossaries)              on input line 1.
```

When used with the tex option, the verbose option will additionally write information while TEX is sorting, since this can take a while and may give the appearance that the build process has hung.

When used with the bib2gls option, the verbose option will show the syntax used by `\tstidxmakegloss` to load each resource. If you search the `.log` file for instances of `\GlsXtrLoadResource`, you'll find the commands needed to replicate the behaviour of `\tstidxmakegloss`.

In addition to the options listed above, the following options are also available for testidx-glossaries:

extra Load the glossaries-extra package.

noextra Don't load the glossaries-extra package. Just load the base glossaries package. (Default.)

makeindex (Default.) Passes the makeindex option to glossaries. This option also sets up `\tstidxmakegloss` to use `\makeglossaries`, `\tstidxprintglossaries` to use `\printglossaries` and `\tstidxprintglossary` to use `\printglossary`. Use `makeglossaries` (or `makeglossaries-lite`) in the build process.

xindy Passes the xindy option to glossaries. This option also sets up `\tstidxmakegloss` to use `\makeglossaries`, `\tstidxprintglossaries` to use `\printglossaries` and `\tstidxprintglossary` to use `\printglossary`. Use `makeglossaries` (or `makeglossaries-lite`) in the build process.

tex This option also sets up `\tstidxmakegloss` to use `\makenoidxglossaries`, `\tstidxprintglossaries` to use `\printnoidxglossaries` and `\tstidxprintglossary` to use `\printnoidxglossary`. (TEX is used for to sort and collate the entries. Don't use `makeglossaries` or `makeglossaries-lite` in the build process.)

bib2gls Passes the record option to glossaries-extra. (This option automatically implements the extra option.) This option also sets up `\tstidxmakegloss` to use `\GlsXtrLoadResources`, `\tstidxprintglossaries` to use `\printunsrtglossaries` and `\tstidxprintglossary` to use `\printunsrtglossary`. Use `bib2gls` in the build process. Note that this option ignores the commands `\tstidxindexmarkerprefix` and `\tstidxmathsymprefix`.

manual Indicates that the test document doesn't use `\tstidxmakegloss`. (This disables the check that ensures that command has been used.) Use this option if you want to customize the glossary set-up. This option may be used in addition to the above options, but it will disable `\tstidxmakegloss`, `\tstidxprintglossary` and `\tstidxprintglossaries`.

The sample files can be loaded using

```
\tstidxloadsamples
```

```
\tstidxloadsamples
```

(which `\tstidxmakegloss` does implicitly) except in the case of bib2gls where the sample files need to be loaded in `\GlsXtrLoadResource`.

seekey (Default.) Use the `see` key for cross-references instead of using `\glssee`. If the `seealso` key has been defined (glossaries-extra v1.16+), then this will be used for the "see also" cross-references (otherwise `see=[\seealsoname]{`⟨*label*⟩`}` will be used).

noseekey Use `\glssee` for cross-references and don't set the `see` (or `seealso`) key.

noglsnumbers Passes the glsnumbers=false option to glossaries.

glsnumbers Passes the glsnumbers=true option to glossaries. (This is the default for the glossaries package.)

desc Provide descriptions for the dummy entries. This setting automatically implements the glossaries package's nopostdot=false option and sets the `indexgroup` glossary style.

nodesc (Default.) Don't provide descriptions for the dummy entries. (The `description` field is set to empty.) This setting automatically implements the glossaries package's nopostdot option and sets the `mcolindexgroup` glossary style. (The glossary-mcols package is automatically loaded.)

Both the `mcolindexgroup` and `indexgroup` styles set the `name` field in `\glstreenamefmt`, which by default uses `\textbf`. This can be redefined as appropriate. You can switch to a different glossary style using `\setglossarystyle{`⟨*style-name*⟩`}`.

3 Basic Commands

This section only covers the basic commands provided by testidx and testidx-glossaries. For more advanced commands, see the documented code.

`\testidx`

```
\testidx[⟨blocks⟩]
```

This is the principle command provided by the testidx package. It generates the predefined dummy text that's interspersed with indexing commands. (The text varies slightly according to the document settings.) There are 16 blocks in total. This number can be accessed through the register:

`\tstidxmaxblocks`

```
\tstidxmaxblocks
```

If the optional argument `[`⟨*blocks*⟩`]` is omitted, all the blocks will be used. Each block starts with a number identifying it. This number prefix is formatted using:

\tstidxprefixblock

```
\tstidxprefixblock{⟨n⟩}
```

where ⟨*n*⟩ is the block number. If you want to suppress the number prefix, just redefine this command to ignore its argument.

By default, the blocks are separated by a paragraph break. If the starred form is used, the blocks are separated by a space. Note that some of the blocks contain paragraph breaks for displayed material. The starred form won't eliminate paragraph breaks *within* the blocks, just those used as separators between the blocks.

If you use testidx-glossaries, you additionally need

\tstidxmakegloss

```
\tstidxmakegloss[⟨options⟩]
```

in the preamble. This loads the files that provide the dummy entries and uses \makeglossaries or \makenoidxglossaries or \GlsXtrLoadResources depending on the package options. The optional argument ⟨*options*⟩ is appended to the optional argument of \GlsXtrLoadResources if the bib2gls package option has been used, otherwise ⟨*options*⟩ is ignored.

To display the glossary, either use

\tstidxprintglossaries

```
\tstidxprintglossaries
```

or

\tstidxprintglossary

```
\tstidxprintglossary{⟨options⟩}
```

where you want the glossary to be displayed. This will use the appropriate command according to the package set up.

The intention of the dummy text is to provide an index that should typically span at least three pages for A4 or letter paper, to allow testing of headers and footers across a double-paged spread and to test the effects of page breaking. Some of the indexing commands intentionally cause warnings from makeindex to test for certain situations. Phrases are indexed as well as just individual words to increase the chances of indexed terms spanning a page break. However, the page dimensions, fonts and other material in the document will obviously alter where the page breaks occur.

You can display only a subset of the blocks using the optional argument, which may be a comma-separated list of block identifiers or hyphen-separated range. Note that some of the blocks contain the start or end of an indexing range. If you only display a subset of the blocks that contains any of these, you need to make sure that you include the blocks that contain matching open and closing ranges (unless you're testing for mis-matched ranges).

The optional argument may be a mixture of individual block identifiers and ranges. Examples:

1. Just display block 6:

   ```
   \testidx[6]
   ```

2. Display blocks 4 to 6:

   ```
   \testidx[4-6]
   ```

3. Display blocks 1, 4 to 6, and the last block:

   ```
   \testidx[1,4-6,\tstidxmaxblocks]
   ```

4. Intersperse the blocks with sections:

   ```
   \section{Sample}
   \testidx[1-6]
   \section{Another Sample}
   \testidx[7-\tstidxmaxblocks]
   ```

If for some bizarre and wacky reason you want the blocks in the reverse order, you can do so. For example:

```
\testidx[\tstidxmaxblocks-1]
```

However the open and close range formations are likely to confuse `makeindex`/`xindy`, but perhaps that's your intention. Just remember to stay within the range 1–`\tstidxmaxblocks` as you'll get an error if you go out of those bounds.

With just testidx, the actual indexing is performed using:

`\tstindex`

```
\tstindex{⟨text⟩}
```

This defaults to just `\index{`⟨*text*⟩`}` but may be redefined. For example, if you are testing multiple indexes, you can redefine `\tstindex` to use a specific index.

With testidx-glossaries, the above command isn't used. Instead `\gls`, `\glspl`, `\glsadd` or `\glssee` will be used depending on the context.

The dummy text includes markers to identify where the instances of `\tstindex` have been used. To reduce the possibility of package conflict, testidx loads a bare minimum of packages[1] and tries to rely as much as possible on LaTeX kernel commands, so the markers are fairly primitive. If you prefer fancier markers, you can change them by redefining the commands listed below. Multiple markers in the dummy text indicate multiple instances of `\tstindex` without any intervening text. (Naturally, testidx-glossaries requires more packages as it loads glossaries, and possibly also glossaries-extra.)

`\tstidxmarker`

[1] only color, ifxetex and ifluatex are loaded

```
\tstidxmarker
```

This is the marker used to show an instance of \tstindex for a top-level entry that doesn't start or end a range. Default: ·

\tstidxopenmarker

```
\tstidxopenmarker
```

This is the marker used to show an instance of \tstindex for a top-level entry that starts a range. Default: ⌈

\tstidxclosemarker

```
\tstidxclosemarker
```

This is the marker used to show an instance of \tstindex for a top-level entry that ends a range. Default: ⌋

\tstidxsubmarker

```
\tstidxsubmarker
```

This is the marker used to show an instance of \tstindex for a sub-entry that doesn't start or end a range. Default: ˬ

\tstidxopensubmarker

```
\tstidxopensubmarker
```

This is the marker used to show an instance of \tstindex for a sub-entry that starts a range. Default: ⌈

\tstidxclosesubmarker

```
\tstidxclosesubmarker
```

This is the marker used to show an instance of \tstindex for a sub-entry that ends a range. Default: ⌋

\tstidxsubsubmarker

```
\tstidxsubsubmarker
```

This is the marker used to show an instance of \tstindex for a sub-sub-entry that doesn't start or end a range. Default: ˬ

\tstidxopensubsubmarker

```
\tstidxopensubsubmarker
```

This is the marker used to show an instance of \tstindex for a sub-sub-entry that starts a range. Default: ⌜

\tstidxclosesubsubmarker

\tstidxclosesubsubmarker

This is the marker used to show an instance of \tstindex for a sub-sub-entry that ends a range. Default: ⌟

\tstidxseemarker

\tstidxseemarker

This is the marker used to show an instance of \tstindex that uses a cross-reference. Additionally, the cross-referenced information will appear in a marginal note. Default: ˆ

\tstidxsubseemarker

\tstidxsubseemarker

This is the marker used to show an instance of \tstindex that uses a cross-reference in a sub-entry. Default: ˆ̌ (the sub-level and cross-reference markers superimposed, not to be confused with a sub-level marker followed by a cross-reference marker, which indicates consecutive occurrences of \tstindex). As above the cross-reference information appears in a marginal note. The main term and the sub-entry term are separated with the symbol given by

\tstidxsubseesep

\tstidxsubseesep

which defaults to ⊳

There are three encap values used:

\tstidxencapi

\tstidxencapi{⟨*location*⟩}

\tstidxencapii

\tstidxencapii{⟨*location*⟩}

\tstidxencapiii

\tstidxencapiii{⟨*location*⟩}

By default these just set ⟨*location*⟩ in a different text colour.

If you are using xindy, you'll need to add these to a .xdy file that can be loaded using xindy's -M switch. For example, with just testidx, include the following in your .xdy file:

```
; 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")
```

You may also want to add the list and range separators, if you haven't already done so:

```
(markup-locref-list :sep ",")
(markup-range :sep "--")
```

If you use testidx-glossaries, the glossaries package provides commands to add information to the automatically generated `.xdy` file. For example:

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

If you want to provide your own custom cross-reference class you can use

`\tstidxSetSeeEncap`

```
\tstidxSetSeeEncap{⟨encap name⟩}
```

to change the `see` encap to ⟨*encap name*⟩ and

`\tstidxSetSeeAlsoEncap`

```
\tstidxSetSeeAlsoEncap{⟨encap name⟩}
```

to change the `seealso` encap to ⟨*encap name*⟩. For example:

```
\tstidxSetSeeAlsoEncap{uncheckedseealso}
```

and in the `.xdy` file:

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

which creates an unverified alternative to `seealso`.

The `\tstindex` command is sometimes placed before the term or phrase being indexed and sometimes afterwards. To clarify what's being indexed, the adjacent word or phrase is surrounded by

`\tstidxtext`

`\tstidxtext{⟨text⟩}`

This defaults to using a dark grey text colour. If an encap has been used, the corresponding encap command (see above) is included within the argument of `\tstidxtext`:

`\tstidxtext{⟨cs⟩{⟨text⟩}}`

where ⟨*cs*⟩ is the encap command. This means that with the default definitions, the dark grey text colour will only be visible when there's no encap, as the encap command will override the colour change.

Note that the marker is included within ⟨*text*⟩. Some of the examples have consecutive uses of `\tstindex`, such as a top-level entry followed by a sub-entry. For example, a person's name is indexed twice:

```
Donald Knuth\index{Knuth, Donald}\index{people!Knuth, Donald}
```

(It's actually done using `\tstidxperson{Donald}{Knuth}` for better consistency. These markup commands typically won't need changing, but if they do, see the documented code for further detail.)

In the case of testidx-glossaries, the above example would be

```
\gls{DonaldKnuth}\glsadd{people.DonaldKnuth}
```

(`\index` isn't used).

Example (using just testidx):

```
\renewcommand*{\tstindex}[1]{}
\textsf{\testidx[1,\tstidxmaxblocks]}
```

This produces the two paragraphs (first and last blocks) shown below:

1. This is a sample block of text designed to test `\index`, the layout of the index (`theindex` environment) and any indexing application, such as `makeindex` or `xindy`. This text is just filler (produced using `\testidx` provided by the `testidx` package) to pad out the document with instances of `\index` interspersed throughout. You can use it, for example, to test an indexing package, such as `makeidx` or `imakeidx`, or to test a `makeindex` style file or `xindy` module. You can find out more information from the

^padding, filler

testidx user manual, which can be accessed using the texdoc application. This block starts a range that is closed in block 16.

16. This is the final block of dummy text provided by the testidx package. This block contains the close of a range that was started in block 1. Fun, wasn't it?

Note that I've redefined \tstindex to ignore its argument in this document so those terms won't actually be indexed in this case. The block references (such as "block 1") in the dummy text don't use the standard \label/\ref mechanism as the references must still work even if the referenced block has been omitted. This means they won't have hyperlinks even if you include the hyperref package as the target may not be defined. They are provided primarily so you can easily find out which blocks need adding if you're only using a subset and need to close a range.

4 Indexing Special Characters

If you need to change the indexing special characters, you can redefine the commands listed in this section. Remember that you will also need to make the relevant changes to your indexing style file. (These commands only apply to testidx not testidx-glossaries.)

\tstidxquote

```
\tstidxquote
```

The "quote" character. The default is: ". Note that the german or ngerman package option will automatically redefine \tstidxquote to + (plus).

\tstidxactual

```
\tstidxactual
```

The "actual" character. The default is: @.

\tstidxlevel

```
\tstidxlevel
```

The "level" character. The default is: !.

\tstidxencap

```
\tstidxencap
```

The "encap" character. The default is: |.

\tstidxopenrange

```
\tstidxopenrange
```

The "open range" character. The default is: (.

\tstidxcloserange

```
\tstidxcloserange
```

The “close range” character. The default is:).

5 Extended Latin Characters

The dummy text includes words or phrases that have extended Latin characters. There are two modes:

ASCII This mode is on *unless* you are using XeLaTeX or LuaLaTeX, or the document has loaded inputenc with the encoding set to utf8.

Example that will switch on ASCII mode:

```
\documentclass{article}

\usepackage[latin1]{inputenc}
\usepackage{makeidx}
\usepackage{testidx}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

UTF-8 This mode is on *if* you are using XeLaTeX or LuaLaTeX, or if the document has loaded inputenc with the encoding set to utf8.

Example that will switch on UTF-8 mode:

```
\documentclass{article}

\usepackage{fontspec}
\usepackage{makeidx}
\usepackage{testidx}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

Or

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage{makeidx}
\usepackage{testidx}

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

If the UTF-8 mode is on, you can additionally use the diglyphs package option to replace the “ll”, “ij” and “dz” digraphs with a single glyph, but you’ll need a font that supports those glyphs. (The trigraph “dzw” and other digraphs, such as “th” aren’t affected by this option.) For example

```
\documentclass{article}

\usepackage{fontspec}
\setmainfont{Linux Libertine O}

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

\makeindex

\begin{document}
\testidx

\printindex
\end{document}
```

When the ASCII mode is on, words or phrases with UTF-8 characters use the standard LaTeX accent commands, such as `\'` (acute accent) or `\o` (ø). There are two package options that determine whether or not to include these commands in the sort key: stripaccents will remove the accent commands (except for the umlaut shortcut `"` if the german or ngerman package option has been used), and nostripaccents will keep the accent commands in the sort key.

For example, with the ASCII mode on with the stripaccents option, “Anders Jonas Ångström” is indexed as

```
Angstrom, Anders Jonas@\AA ngstr\""om, Anders Jonas
```

unless the german or ngerman option is on, in which case it's indexed as

```
Angstr"om, Anders Jonas@\AA ngstr"om, Anders Jonas
```

Whereas with the nostripaccents option, this name is indexed as

```
\r Angstr\""om, Anders Jonas@\AA ngstr\""om, Anders Jonas
```

unless the german or ngerman option is on, in which case it's indexed as

```
\r Angstr"om, Anders Jonas@\AA ngstr"om, Anders Jonas
```

When the UTF-8 mode is on, UTF-8 characters are used instead. For example, "Anders Jonas Ångström" is indexed as

```
Ångström, Anders Jonas
```

(The stripaccents and nostripaccents options are ignored.)

XeLaTeX and LuaLaTeX both natively support UTF-8, so when either of those engines are in use, the UTF-8 characters will be written to the indexing file as they are. So the above example will appear in the `.idx` file as:

```
\indexentry{Ångström, Anders Jonas}{⟨location⟩}
```

Regular LaTeX (`latex` or `pdflatex`) requires the inputenc package to support UTF-8 characters, but each UTF-8 character is treated as two tokens (the first and second octets) where the first token is an active character that takes the second token as the argument. This means that expansion will occur when writing these active characters to an external file. This means that the above will appear in the `.idx` file as:

```
\indexentry{\IeC {\r A}ngstr\IeC {\"o}m, Anders Jonas}{3}
```

(where 3 is the page number).

Since this can confuse the indexing application, testidx provides a sanitize package option which will first sanitize the UTF-8 characters before indexing them. This option is on by default for regular LaTeX and off for XeLaTeX and LuaLaTeX. You can switch it off using the nosanitize package option.

Whether it should be on or off really depends on what you want to test. For example, if you want to test how an indexing application deals with UTF-8 characters, then switch it on, but if you want to test how your indexing command (whatever `\tstindex` is defined as) behaves with these characters, then switch it off.

Note that this sanitize option isn't adjusting the definition of `\index` or `\tstindex`, but is essentially pretending that the user is doing something like:

```
\makeatletter
Anders Jonas Ångström%
\def\tmp{Ångström, Anders Jonas}%
\@onelevel@sanitize\tmp
\exandafter\index\expandafter{\tmp}%
\edef\tmp{people\tstidxlevel\tmp}%
\exandafter\index\expandafter{\tmp}%
```

instead of simulating:

```
Anders Jonas Ångström%
\tstindex{Ångström, Anders Jonas}%
\tstindex{people!Ångström, Anders Jonas}%
```

Note that the sanitization isn't applied to the entire argument of `\tstindex`, but only selected parts of it.

Index

X