

X_ƒIndex

v.0.2

2010/01/17

Paul Isambert

zappathustra@free.fr

Abstract

X_ƒIndex is a package based on X_ƒSearch that automatically indexes words in a X_ƒL^AT_EX document. Words or phrases (possibly underspecified) are declared in a list and each of their occurrences then creates an index entry, whose content might be freely specified beforehand.

Automatic indexes are bad. You know that. Hence the severe look of this documentation. So: don't use X_ƒIndex. Or: use it as a tool to generate an index whose relevance you then check. Or: use it for entries that are generally indexed on every occurrence, like proper names. Or: do bad indexes.

You load X_ƒIndex in the usual way:

```
\usepackage{xƒindex}
```

There's only one package option, namely `mark`, which prefixes all index entries generated by X_ƒIndex with `***[⟨word⟩:⟨line number⟩]`, where `⟨word⟩` is the word that generated the entry and `⟨line number⟩` the line where you can find it in the `.tex` file. Moreover, if a word appears more than once on a page, and thus generates several index entries although they'll be merged in the typeset index, with this option each entry is listed along with each occurrence of the word. So these entries end up at the beginning of the index, sorted with symbols, apart from the entries generated by the usual `\index` command. Thus they can be easily checked and prevented if irrelevant.

```
\makeindex XƒIndex loads the makeidx package, so \makeindex and \printindex, which should be executed as  
\printindex usual, as well as the usual \index command, are available. (If you don't know what I'm talking  
\index{⟨entry⟩} about, then you don't know how to produce an index. You should read the makeidx documentation  
at least, or the rest of this document might seem cryptic.)
```

```
\IndexList{*}{⟨name⟩}{⟨list of entries⟩} This is XƒIndex's main command. The star is optional and ⟨list of entries⟩ is made of ⟨word⟩=⟨entry⟩  
pairs separated by commas, with the part in red optional too. Let's see the simplest possibility first:
```

```
\IndexList{mylist}{alley cat,dog,gnu}
```

This will index *alley cat* on every occurrence of *alley cat* in your document, *dog* on every occurrence of *dog*, and *gnu* on every occurrence of *gnu*. But this will also index *alley cat* when seeing *Alley cat*, *dog*

when seeing *dOg*, and *gnu* when seeing *GNU*. In other words, `XqIndex` is case-insensitive by default, and the index entries are put in lower case. Most of the time, that's useful. But a *gnu* is not *GNU*, so sometimes you want a different behavior. In this case, put a `*` before the word, e.g.:

```
\IndexList{mylist}{alley cat,DOG,gnu,*GNU}
```

Now `XqIndex` will index *gnu* when seeing *gnu* or *Gnu*, but it will index *GNU* when seeing *GNU*. Note that `DOG` without a star will index *dog* in any case display, and the index entry will be *dog*, not *DOG*. You can also make a whole list case-sensitive, by adding the star just after `\IndexList`, as in:

```
\IndexList{mylist}{alley cat,dog,gnu}
\IndexList*{mylist}{GNU}
```

All words in a `*`-marked list are case-sensitive, and you should not add another star before them. Note that you can use the same list with different case-sensitivities each time, as in the above example.

Case-sensitivity is useful mostly to index proper names.

The words in a list need not be fully specified. You can index all words beginning or ending with some letters. To do so, use `?` for the unspecified part. For instance,

```
\IndexList{mylist}{cat?,?mals,*?NU}
```

will find and index *cat*, *cats*, *catheter*, *animals*, *mammals*, *GNU*, *gNU*, but not *gnu*, because `*?NU` means ‘a word that ends with “NU” in upper case.’ Each of these words will be indexed in its own entry, so there'll be a *cat* and a *cats* entry, but we'll learn how to have them grouped presently. If a word matches several underspecified forms, the more specific wins, e.g. *mammals* matches `mam?` and not `ma?`, and ‘starting-with’ forms always win against ‘ending-with’ ones, no matter the degree of specificity, e.g. *cat* matches `c?` and not `?at`. Phrases, i.e. words separated by blanks (like *alley cat* above) can be underspecified only at the end, i.e. you can say `alley ca?` but not `?ley cat` (actually you can try but it won't work). Finally, underspecified parts are only the beginning or the end of a word, i.e. you can't say `c?t`, and there should be only one underspecification, i.e. `?a?` is forbidden.

Now we come to the interesting part in $\langle word \rangle = \langle entry \rangle$, namely the red part: $\langle entry \rangle$ should be a pseudo-index entry, as it were, i.e. it can be a normal index entry, as in

```
\IndexList{another list}{%
dog=mammal|textit,
snail=Obviously Not Mammal,
cat=mammal!pet,
```

```
horse?=horse@{\bfseries horse}}
```

Here, every occurrence of *dog* will produce an index entry at *mammal* with the page number in italic, *snail* will index *Obviously Not Mammal*, *cat* will create a sub-entry *pet* in the *mammal* entry and *horse*, *horses*, *horseshoe*, etc., will create a bold entry at *horse*. So, as you can see, the left part of the expression specifies which words should generate an index entry, and the right part is the index entry that should be generated. The latter is not case-insensitive anymore: although `snail` is case-insensitive and will fire on *snail*, *Snail* or *SNaIl*, the index entry itself will be as specified, i.e. *Obviously Not Mammal* and not *obviously not mammal*.

Now you can index variations of a word under the same entry. As I've just said, *horse* and *horses* will both be indexed under *horse*, as well as all their case-variants, as we already know. (*Horseshoe* will go in that entry too, so beware).

The right part of those pairs is a 'pseudo-index entry,' and not a proper index entry, because the word to index can be omitted, in which case the word that fires the index entry will be used instead. More precisely, whenever `XgIndex` encounters one of the usual `MakeIndex` operators, namely `!` (for a sub-entry), `|` (for a control sequence) and `@` (to indicate the form of the entry irrespective of alphabetical order), at the beginning of an index entry, it reinterprets this entry with the the firing word or phrase on its left. I.e.

```
\IndexList{yet another list}{%
  dog=@dog (canis lupus familiaris),
  cat=!basic definition,
  horse?={\textbf}}
```

indexes every occurrence of *dog* as *dog (canis lupus familiaris)*, but with the alphabetical order of *dog*, every occurrence of *cat* as a *basic definition* subentry in the *cat* entry, and every occurrence of a word beginning with *horse* as an entry for this word with the page number in boldface. Remember that the `\index` command is still available, so you can still index *cat* as *cat* in your document.

It's obviously a very bad idea to say something like `\IndexList{mylist}{cat=|{}}`, since this will fire `\index{cat|{}}` on every occurrence of *cat*. So page ranges can't be declared by `XgIndex` (it would be a very bad idea anyway). On the other hand, `MakeIndex` automatically creates page ranges as soon as an entry is found on at least three successive pages, unless you run it with the `-r` option.

If you want a comma in the right part of the entry, enclose the entire entry, minus the `MakeIndex` operator if any, between braces, e.g.:

```
\IndexList{writers}{kafka={Kafka, Franz}}
```

(This did not work in version 0.1, and now it's corrected thanks to Simon Spiegel who indicated it to me.)

`\StopIndexList{<lists>}` Here are some additional macros to let you regulate the flow of the indexing frenzy. `\StopIndexList` takes a comma-separated list of lists and turn them off. `\StopIndex` turns off all lists. `\NoIndex` simply prevents *<text>* from being indexed. It's very important, because it lets you prevent irrelevant indexation in the body of your document.

That's all you need to know to use `XqIndex`. The next paragraph describes how `XqIndex` sets the parameters of `XqSearch`; so if you don't know `XqSearch` and don't intend to use it, there's no need to read what follows.

`XqIndex` keeps `XqSearch`'s default search order, namely full words before prefixes before suffixes, with case-sensitive tests first each time. Affixes are modified, however: they're sorted by length (longer ones first) and not kept in the order they were declared, and only one affix fires in case of a successful test, instead of all the affixes of a given test. You can modify these specifications since `XqIndex` uses !-marked replacement texts, so they won't embed each other, but then you might end up with multiple entries and a lack of consistency.

You can use `\StartSearching` and `\StopSearching` instead of `\StopIndex`, which for the moment renders all lists unavailable. The former two commands, however, will stop all lists defined by `XqSearch`.

The default set of boundaries is left untouched, i.e. its members are: ., ; :- ' ' O [] { }

Implementation

Basic declarations and definitions.

```
1 \ProvidesPackage{!FileName}![!FileDate !FileVersion Automatic index for xqLaTeX.]
2 \RequirePackage{makeidx,xqSearch}
3 \makeatletter
4 \newif\ifxqi@mark
5 \DeclareOption{mark}{\xqi@marktrue}
6 \ProcessOptions
```

```
\xqi@Mark \xqi@Mark either shows the word and the corresponding line, or it gobbles it. It is placed at the begin-
\xqi@empty ning of the \index command, whose expansion is delayed accordingly.
\xqi@end 7 \ifxqi@mark
\xqi@Lists 8 \def\xqi@Mark#1{***[#1:\the\inputlineno] }
```

```

9 \else
10 \def\xi@Mark#1{}
11 \fi
12 \def\xi@empty{}
13 \def\xi@end{\xi@end}
14 \def\xi@Lists{}

```

`\IndexList` Most of the job is done by `XqSearch`. What we need to do is properly analyze the entry to launch an
`\xi@IndexList` adequate search.

```

15 \def\IndexList{%
16 \ifstar{\def\xi@cs{*}\xi@IndexList}{\def\xi@cs{} \xi@IndexList}%
17 }
18 \def\xi@IndexList#1#2{%
19 \def\xi@ListName{#1}%
20 \edef\xi@Lists{\xi@Lists#1,}%
21 \unless\ifcsname#1@xeindex\endcsname
22 \csname#1@xeindex\endcsname

```

When a index list is created, we associate five `XqSearch` lists with it: one is for words and affixes that should index themselves in lower case.

```

23 \SearchList{#1@xeindex@ncs@normal@list}{%
24 \def\xi@word{##1}%
25 \lowercase{\expandafter\index\expandafter{\xi@Mark\xi@word##1}}}{%

```

Another one is for case-sensitive words and affixes.

```

26 \SearchList{#1@xeindex@cs@normal@list}{%
27 \expandafter\index\expandafter{\xi@Mark{##1}##1}}}{%

```

And the last three are for words and affixes that launch a special entry, which is stored in an associated command.

```

28 \SearchList{#1@xeindex@ncs@special@list}{%
29 \lowercase{\csname##1@#1@xeindex@entry\endcsname}{##1}}}{%
30 \SearchList{#1@xeindex@cs@special@list}{%
31 \csname##1@#1@xeindex@entry\endcsname{##1}}}{%
32 \SearchList{#1@xeindex@affix@special@list}{%
33 \csname\AffixFound @#1@xeindex@entry\endcsname{##1}}}{%
34 \fi
35 \xi@ParseList#2,\xi@end,%

```

```

36 }
\Xi@ParseList This macro recursively tests each entry in \SearchList and feed it to \Xi@ParseEntry with an additional = to check for the right part. It also adds \Xi@cs, which was defined do * in case \SearchList was starred.
37 \def\Xi@ParseList#1,{%
38   \def\Xi@temp{#1}%
39   \ifx\Xi@temp\Xi@end
40     \let\Xi@next\relax
41   \else
42     \expandafter\Xi@ParseEntry\Xi@cs#1=\Xi@end
43     \let\Xi@next\Xi@ParseList
44   \fi\Xi@next
45 }
\Xi@ParseEntry This one analyses the entry. If the third argument is empty, then there is no =⟨entry⟩ part in the entry. In this case we add the word or affix to one of the two simple lists, depending on its case-sensitivity.
46 \def\Xi@ParseEntry#1#2=#3\Xi@end{%
47   \def\Xi@temp{#3}%
48   \ifx\Xi@temp\Xi@empty
49     \expandafter\if\noexpand#1*%
50       \AddToList!\Xi@ListName @xeindex@cs@normal@list}{#1#2}%
51   \else
52     \AddToList!\Xi@ListName @xeindex@ncs@normal@list}{#1#2}%
53   \fi
Otherwise we feed the right part to \Xi@MakeEntry which sets the \ifXi@Noword switch. We also check whether the word is an affix or not, and whether it is case-sensitive. The word is then associated to the right list and an associated macro is created.
54 \else
55   \Xi@MakeEntry#3%
56   \expandafter\if\noexpand#1*%
57     \Xi@CheckAffix#2?\Xi@end
58     \ifXi@Affix
59       \AddToList!\Xi@ListName @xeindex@affix@special@list}{#1#2}%
60     \expandafter\edef\csname\Xi@Affix @\Xi@ListName @xeindex@entry\endcsname##1{%

```

```

61         \unexpanded{\expandafter\index\expandafter}{%
62         \noexpand\xi@Mark{##1}\ifxi@Noword##1\fi\unexpanded\expandafter{\xi@temp}}%
63     }%
64 \else
65     \AddToList!\xi@ListName @xeindex@cs@special@list}{#1#2}%
66     \expandafter\edef\csname#2@\xi@ListName @xeindex@entry\endcsname##1{%
67     \unexpanded{\expandafter\index\expandafter}{%
68     \noexpand\xi@Mark{##1}\ifxi@Noword#2\fi\unexpanded\expandafter{\xi@temp}}%
69     }%
70 \fi
71 \else
72     \xi@CheckAffix#1#2?\xi@end
73     \ifxi@Affix
74         \AddToList!\xi@ListName @xeindex@affix@special@list}{#1#2}%
75         \expandafter\edef\csname\xi@lCAffix @\xi@ListName @xeindex@entry\endcsname##1{%
76         \unexpanded{\def\xi@Word}{##1}%
77         \noexpand\lowercase{
78         \unexpanded{\expandafter\index\expandafter}{%
79         \unexpanded{\xi@Mark{\xi@Word}}%
80         \ifxi@Noword##1\fi\unexpanded\expandafter{\xi@temp}}%
81         }%
82     }%
83 \else
84     \AddToList!\xi@ListName @xeindex@ncs@special@list}{#1#2}%
85     \lowercase{%
86     \expandafter\edef\csname#1#2@\xi@ListName @xeindex@entry\endcsname##1{%
87     \unexpanded{\def\xi@Word}{##1}%
88     \unexpanded{\expandafter\index\expandafter}{%
89     \unexpanded{\xi@Mark{\xi@Word}}%
90     \ifxi@Noword#1#2\fi\unexpanded\expandafter{\xi@temp}}%
91     }%
92     }%
93 \fi
94 \fi

```

```
95 \fi
96 }
```

`\xi@MakeEntry` This determines whether the entry starts with one of the MakeIndex operators.

```
97 \newif\ifxi@Noword
98 \def\xi@exclam{!} \def\xi@at{@} \def\xi@bar{|}
99 \def\xi@MakeEntry#1#2={%
100 \def\xi@temp{#1#2}%
101 \xi@Nowordtrue
102 \unless\ifx\xi@temp\xi@exclam
103 \unless\ifx\xi@temp\xi@at
104 \unless\ifx\xi@temp\xi@bar
105 \xi@Nowordfalse
106 \fi
107 \fi
108 \fi
109 }
```

`\xi@CheckAffix` If the first argument is ?, then the word is unspecified at the beginning. Otherwise, if the third argument is not empty, then it is unspecified at the end (because we added a ? when giving the word to this macro). In case the ? is misplaced, X_qSearch will detect it later.

```
110 \newif\ifxi@Affix
111 \def\xi@CheckAffix#1#2?#3\xi@end{%
112 \xi@Affixfalse
113 \expandafter\if\noexpand#1?%
114 \xi@Affixtrue
115 \def\xi@Affix{#2}%
116 \lowercase{\def\xi@lcAffix{#2}}%
117 \else
118 \def\xi@@temp{#3}%
119 \unless\ifx\xi@@temp\xi@empty
120 \xi@Affixtrue
121 \def\xi@Affix{#1#2}%
122 \lowercase{\def\xi@lcAffix{#1#2}}%
123 \fi
124 \fi
```



```

125 }
\StopIndexList These are straightforward.
\xi@StopIndexList 126 \def\StopIndexList#1{%
\StopIndex 127 \xi@StopIndexList#1,\xi@end,%
\NoIndex 128 }%
129 \def\xi@StopIndexList#1,{%
130 \def\xi@temp{#1}%
131 \ifx\xi@temp\xi@end
132 \let\xi@next\relax
133 \else
134 \StopList{%
135 #1@xeindex@ncs@normal@list,%
136 #1@xeindex@cs@normal@list,%
137 #1@xeindex@cs@normal@list,%
138 #1@xeindex@ncs@special@list,%
139 #1@xeindex@cs@special@list,%
140 #1@xeindex@affix@special@list%
141 }%
142 \let\xi@next\xi@StopIndexList
143 \fi\xi@next
144 }
145 \def\StopIndex{%
146 \expandafter\xi@StopIndexList\xi@Lists\xi@end,%
147 }
148 \def\NoIndex#1{%
149 \bgroup
150 \StopIndex
151 #1%
152 \egroup
153 }
\xi@PrintIndex Finally, we patch \printindex so it won't be searched, and sets XqSearch's parameters.
154 \let\xi@PrintIndex\printindex \def\printindex{\StopIndex\xi@PrintIndex}
155 \SortByLength{pPSS} \SearchOnlyOne{pPSS}
156 \makeatother

```