in-dexter

An index package for Typst

Version 0.3.0 (11. Mai 2024)

Rolf Bremer, Jutta Klebe

Contributors: @epsilonhalbe, @sbatial

1 Sample Document to Demonstrate the in-dexter package

Using the in-dexter package in a typst document consists of some simple steps:

1. Importing the package `in-dexter`.
2. Marking the words or phrases to include in an index.
3. Generating the index page(s) by calling the `make-index()` function.

1.1 Importing the Package

The in-dexter package is currently available on GitHub in its home repository (https://github.com/RolfBremer/in-dexter). It is still in development and may have breaking changes in its next iteration.

```
#import "./in-dexter.typ": *
```

The package is also available via Typst's build-in Package Manager:

```
#import "@preview/in-dexter:0.3.0": *
```

Note, that the version number of the typst package has to be adapted to get the wanted version.

1.2 Marking of Entries

We have marked several words to be included in an index page. The markup for the entry stays invisible . Its location in the text gets recorded, and later it is shown as a page reference in the index page.

```
#index[The Entry Phrase]
```

or

```
#index([The Entry Phrase])
```

or

```
#index("The Entry Phrase")
```

Entries marked this way are going to the "Default" Index. If only one index is needed, this is the only way needed to mark entries. In-dexter can support muiltiple Indexes. To specify the target index for a marking, the index must be addressed.

```
#index(index: "Secondary")[The Entry Phrase]
```

This is the explicit adressing of the secondary index. It may be useful to define a function for the alternate index, to avoid the explicitness:

```
#let index2 = index.with(index:"Secondary")

#index2[One Entry Phrase]
#index2[Another Entry Phrase]
```

1.3 Advanced entries

1.3.1 Symbols

Symbols can be indexed to be sorted under `"Symbols"`, and be sorted at the top of the index like this

```
#index(initial: (letter: "Symbols", sorty-by: "#"), [$(rho)$])
```

1.3.2 Nested entries

Entries can be nested. The `index` function takes multiple arguments - one for each nesting level.

```
#index("Sample", "medical", "blood")
#index("Sample", "medical", "tissue")
#index("Sample", "musical", "piano")
```

1.3.2.1 LaTeX Style index grouping

Alternatively or complementing to the grouping syntax above, the “bang style” syntax known from LaTeX can be used:

```
#index("Sample!mediacal!X-Ray")
```

They can even be combined:

```
#index("Combigroup", "Sample!musical!Chess!")
```

Note that the last bang is not handled as a separator, but is part of the entry. To use the bang grouping syntax, the `make-index()` function must be called with the parameter `use-bang-grouping: true`:

```
#make-index(use-bang-grouping: true)
```

1.3.3 Skipping physical pages

If page number 1 is not the first physical page of the document, the parameter `use-page-counter` of the make-index() function can be set to `true`. Default is `false`.

1.3.4 Formatting Entries

Entries can be formatted with arbitrary functions that map `content` to `content`

```
#index(fmt: it => strong(it), [The Entry Phrase])
```

or shorter

```
#index(fmt: strong, [The Entry Phrase])
```

For convenience in-dexter exposes `index-main` which formats the entry bold. It is semantically named to decouple the markup from the actual style. One can decide to have the main entries slanted or color formatted, which makes it clear that the style should not be part of the function name in markup. Naming markup functions according to their purpose (semantically) also eases the burden on the author, because she must not remember the currently valid styles for her intent.

Another reason to use semantically markup functions is to have them defined in a central place. Changing the style becomes very easy this way.

```
#index-main[The Entry Phrase]
```

It is predefined in in-dexter like this:

```
#let index-main = index.with(fmt: strong)
```

Here we define another semantical index marker, which adds an “ff.” to the page number.

```
#let index-ff = index.with(fmt: it => [#it _ff._])
```

1.4 The Index Page

To actually create the index page, the `make-index()` function has to be called. Of course, it can be embedded into an appropriately formatted environment , like this:

```
#columns(3)[
    #make-index()
]
```

The `make-index()` function accepts an optional array of indexes to include into the index page. The default value, `auto`, takes all entries from all indexes. The following sample only uses the entries of the secondary and tertiary index. See sample output in Section 6.

```
#columns(3)[
    #make-index(indexes: ("Secondary", "Tertiary"))
]
```

2 Why Having an Index in Times of Search Functionality?

A *hand-picked* or *handcrafted* Index in times of search functionality seems a bit old-fashioned at the first glance. But such an index allows the author to direct the reader, who is looking for a specific topic (using index-main), to exactly the right places.

Especially in larger documents and books this becomes very useful, since search engines may provide too many locations of specific words. The index is much more comprehensive, assuming that the author has its content selected well. Authors know best where a certain topic is explained thoroughly or merely noteworthy mentioned (using the `index` function).

Note, that this document is not necessarily a good example of the index. Here we just need to have as many index entries as possible to demonstrate (using a custom made `index-ff` function) the functionality and have a properly filled index at the end.

Even for symbols like `(ρ)`. Indexing should work for for any Unicode string like Cyrillic (Скороспелка) or German (Ölrückstoßabdämpfung). - though we need to add initials `#index(initial: (letter: "C", sort-by: "Ss"), "Скороспелка")` or `#index(initial: (letter: "Ö", sort-by: "Oo"), "Ölrückstoßabdämpfung")`.

---

3 Index

Here we generate the Index page in three columns. The default behavior (auto) is to use all indexes together.

SYMBOLS

(ρ) 3

A

Authors responsibility 3

B

Books 3
Breaking Changes 1

C

Combigroup
  Sample
    Musical
      Chess! 2
Comprehensive 3
Content 3

D

Demonstrate 3 *ff.*
Development 1

E

Engines 3
Entries 3
Environment 2
Example 3
Explained 3

F

Filled 3
Formatting 2
Formatting Entries **2**
Functionality 3

H

Hand Picked 3
Handcrafted 3

I

Index 3
Index Page 1, 2
Invisible 1
Iteration 1

L

Large Documents 3

N

Noteworthy 3

O

Old-fashioned 3

Ö

Ölrückstoßabdämpfung 3

P

Properly 3
Provide 3

S

Sample
  Medical
    Blood 2
    Tissue 2
    X-Ray 2
  Musical
    Piano 2
Search Engines 3
Search Functionality 3
Searching vs. Index **3**
Specific 3

C

Скороспелка 3

T

Thoroughly 3
Topic
  Certain 3
  Specific **3**

4 Secondary Index:

Here we select explicitly only the secondary index.

E

Example	3

S

Specific	3

---

5 Tertiary Index

Here we select explicitly only the tertiary index.

E

Engines	3

F

Filled	3

---

6 Combined Index

Here we select explicitly secondary and tertiary indexes.

E

Engines	3
Example	3

F

Filled	3

S

Specific	3

---

7 Combined Index - all lower case

Here we select explicitly secondary and tertiary indexes and format them all lower case.

E

engines	3
example	3

F

filled	3

S

specific	3