Package

listofitems

v1.52

13 January 2018

Christian Tellechea*
Steven B. Segletes†

This simple package is designed to read a list of items whose parsing separator may be selected by the user. Once the list is read, its items are stored in a structure that behaves as a dimensioned array. As such, it becomes very easy to access an item in the list by its number. For example, if the list is stored in the macro `\foo`, the item number 3 is designated by `\foo[3]`.
A component may, in turn, be a list with a parsing delimiter different from the parent list, paving the way for nesting and employing a syntax reminiscent of an array of several dimensions of the type `\foo[3,2]` to access the item number 2 of the list contained within the item number 3 of the top-tier list.

*unbonpetit@netc.fr
†steven.b.segletes.civ@mail.mil

1 Preface

This package loads no external packages, must be used with the ε-TeX engine, and must be called in (pdf)(Xe)(lua)LaTeX with the invocation

```
\usepackage{listofitems}
```

and under (pdf)(Xe)(Lua)TeX by way of

```
\input listofitems.tex
```

2 Read a Simple List

Set the parsing separator The default parsing separator is the comma and if we want change it, we must do so before reading a list of items, with the definition `\setsepchar{`⟨*parsing-separator*⟩`}`. A ⟨*parsing-separator*⟩ is a set of tokens which possess catcodes different from 1 and 2 (the opening and closing braces), 14 (usually `%`) and 15. The token of catcode 6 (usually `#`) is accepted only if it is followed by an integer, denoting the argument of a macro; In no case should this token be provided alone as the ⟨*parsing-separator*⟩. Commands can be included in this set of tokens, including the TeX primitive `\par`.

The parsing-separator ⟨*delimiter*⟩ "`/`" is reserved by default for nested lists (see page 3). It is therefore not proper to write "`\setsepchar{/}`" because the listofitems package would misunderstand that you want to read a nested list. To set "`/`" as the ⟨*parsing-separator*⟩ for a simple list, it is necessary, using the optional argument, to choose a different parsing-separator ⟨*delimiter*⟩ for nested lists, for example "`.`", and write "`\setsepchar[.]{/}`".

It is not possible to select `|` as the ⟨*delimiter*⟩ because it would conflict with the logical **OR**, denoted "`||`" (see below). However, one can work around this limitation, at one's own peril, writing "`\setsepchar{{|}}`".

Read a list To read the list of items, the `\readlist`⟨*macro-list*⟩`{`⟨*list*⟩`}` should be called. In so doing, the ⟨*list*⟩ is read and the items are stored in a macro, denoted ⟨*macro-list*⟩ which therefore acts as a table with the items of the ⟨*list*⟩. If braces appear as part of a list item, they *must* be balanced. Tokens possessing the catcodes 6, 14 and 15 are not allowed in the lists.
For example, to set the ⟨*macro-list*⟩ named `\foo`, we can write

```
\setsepchar{,}
\readlist\foo{12,abc,x y ,{\bfseries z},,\TeX,,!}
```

If the ⟨*list*⟩ is contained in a macro, then this macro is expanded. Therefore, we can simply employ the syntax `\readlist`⟨*macro-list*⟩⟨*macro*⟩ as in

```
\setsepchar{,}
\def\List{12,abc,x y ,{\bfseries z},,\TeX,,!}
\readlist\foo\List
```

The macro `\greadlist` makes *global* assignments and therefore, enables the use of ⟨*macro-list*⟩ outside of the group where `\greadlist` has been executed.

Access an item The macro `\foo` *requires* a numeric argument in square brackets, which we symbolically denote as *i*, indicating the rank of the item you wish to access. So `\foo[1]` is[3] "`12`". Similarly, `\foo[4]` is "`{\bfseries z}`".
The number *i* can also be negative in which case the counting is done from the end of the list: −1 represents the last item, −2 the penultimate, etc. If the number of items is *n*, then the argument −*n* is the first item.

[3] `\foo[`*i*`]` requires 2 expansions to give the item.

In general, if a ⟨*list*⟩ has a length *n*, then the index *i* can be in the interval [1 ; *n*] or [−*n* ; −1]. Otherwise, a compilation error occurs.
If the index is empty, `\foo[]` produces the complete ⟨*list*⟩.
The macro `\foosep` is created. It is used with the syntax `\foosep[`⟨*index*⟩`]` and allows access to the parsing-separator that follows the item of rank ⟨*index*⟩. The last parsing-separator (the one following the last item) is empty. If the ⟨*index*⟩ is empty, `\foosep[]` is empty.

Select several possible parsing separators To specify several possible separators, use the **OR** operator, denoted "`||`". One can use this feature, for example, to isolate the terms in an algebraic sum:

```
\setsepchar{+||-}
\readlist\term{17-8+4-11}
1) \term[1] (parsing separator = \termsep[1])\par
2) \term[2] (parsing separator = \termsep[2])\par
3) \term[3] (parsing separator = \termsep[3])\par
4) \term[4] (parsing separator = \termsep[4])
```

1) 17 (parsing separator = -)
2) 8 (parsing separator = +)
3) 4 (parsing separator = -)
4) 11 (parsing separator =)

Number of items If we write `\readlist`⟨*macro-list*⟩`{`⟨*list*⟩`}`, then the macro ⟨*macro-list*⟩`len` contains[4] the number of the items in ⟨*list*⟩. In the example with `\foo`, the macro `\foolen` expands to 8.

View all items For purposes of debugging, the macro `\showitems`⟨*macro-list*⟩ includes all items from a list, while the star version displays these items "detokenized." [5]

```
\showitems\foo\par
\showitems*\foo
```

12 abc x y z ‖ TEX ‖ !
12 abc x y {\bfseries z} ‖ \TeX ‖ !

The presentation of each list item is assigned to the macro `\showitemsmacro` whose code is

```
\newcommand\showitemsmacro[1]{%
    \begingroup\fboxsep=0.25pt \fboxrule=0.5pt \fbox{\strut#1}\endgroup
    \hskip0.25em\relax}
```

It is therefore possible — and desirable — to redefine it if we desire a different presentation effect.

The macro `\fbox` and associated dimensions `\fboxsep` and `\fboxrule`, are defined by listofitems, when *not* compiled under LaTeX, to achieve the same result *as if* performed under LaTeX.

Suppression of extreme (leading/trailing) spaces By default, listofitems reads and retains the spaces located at the beginning and end of an item. For these spaces to be ignored when reading the ⟨*list*⟩, execute the starred version `\readlist*`⟨*macro*⟩`{`⟨*list*⟩`}`:

```
\setsepchar{,}
\readlist*\foo{12,abc, x y ,{\bfseries z}, ,\TeX,,!}
\showitems\foo
```

12 abc x y z ‖ TEX ‖ !

Managing empty items By default, the listofitems package retains and accounts for empty items. Thus, in the previous example, the 2nd expansion of `\foo[7]` is empty. For empty items of the list (i.e., those list items defined by two consecutive parsing delimiters) to be ignored, we must, before invoking `\readlist`, execute the macro `\ignoreemptyitems` . To return to the default package behavior, simply execute the macro `\reademptyitems`.

[4] That is to say, it is purely expandable and grows into a number.
[5] The primitive `\detokenize`, conducting this decomposition, inserts a space after each control sequence.

This option can be used alone or in combination with \readlist*, in which case the suppression of extreme (leading/trailing) spaces occurs *before* listofitems ignores the empty list items:

```
\setsepchar{,}
\ignoreemptyitems
\readlist\foo{12,abc, x y ,{\bfseries z}, ,\TeX,,!}
a) number of items = \foolen\par
   \showitems\foo

\readlist*\foo{12,abc, x y ,{\bfseries z}, ,\TeX,,!}
b) number of items = \foolen\par
   \showitems\foo
```

a) number of items = 7
[12] [abc] [x y] [**z**] [ ] [TEX] [!]
b) number of items = 6
[12] [abc] [x y] [**z**] [TEX] [!]

Iterate over a list Once a list read by \readlist and stored in a ⟨*macro-list*⟩, one may iterate over the list with the syntax \foreachitem ⟨*variable*⟩ \in ⟨*macro-list*⟩{⟨*code*⟩}: The ⟨*variable*⟩ is a macro chosen by the user that will loop over the value of each item in the list.
The macro ⟨*variable*⟩cnt represents the sequence number of the item in ⟨*variable*⟩.

```
\setsepchar{ }% parsing-separator = space
\readlist\phrase{One phrase to test.}
\foreachitem\word\in\phrase{List item number \wordcnt{}:  \word\par}
```

List item number 1: One
List item number 2: phrase
List item number 3: to
List item number 4: test.

Assign an item to a macro The \itemtomacro⟨*macro-list*⟩[index]⟨*macro*⟩ assigns to the ⟨*macro*⟩ the item designated by ⟨*macro-list*⟩[index]. The ⟨*macro*⟩ thus defined is purely expandable provided that the tokens in the items are expandable.

```
\setsepchar{ }% parsing-separator = space
\readlist\phrase{One phrase to test.}
\itemtomacro\phrase[2]\aword
\meaning\aword\par
\itemtomacro\phrase[-1]\wordattheend
\meaning\wordattheend
```

macro:->phrase
macro:->test.

3 Nested Lists

We speak of a list being "nested" when asking listofitems to read a list where the items are, in turn, understood as being a list (implying a parsing separator different from the top-tier list). The nesting depth is not limited, but in practice, a depth of 2 or 3 will usually suffice.

Defining the parsing separators To indicate that a list will be nested, so that the list parsing will be performed recursively, one must specify multiple parsing separators, each corresponding to the particular tier of nesting. This list of parsing separators is itself given as a delimited list to the macro \setsepchar, with the syntax \setsepchar[⟨*delimiter*⟩]{⟨*delimited-list-of-parsing-separators*⟩}.
By default, the ⟨*delimiter*⟩ is "/". Thus, writing

```
\setsepchar{\\/,/ }
```

indicates a recursive depth of 3, with the parsing-separator list delimiter defaulting to "/":

— Tier 1 items are parsed between "\\" delimiters;
— Tier 2 items are found within Tier 1 items, parsed between ",", delimiters;
— finally, the Tier 3 items are found within Tier 2 items, parsed between the "␣" delimiters.

The ⟨*depth*⟩ of nesting is contained in the purely expandable macro \nestdepth.

Read and access list items For nested lists, the use of indices obey the following rules:

— `[]` is the main list, i.e., the argument of `\readlist`;
— `[`⟨*i*⟩`]` means the item number ⟨*i*⟩ of the main list;
— `[`⟨*i*⟩`,`⟨*j*⟩`]` means the item number ⟨*j*⟩ of the list mentioned in the previous point (a subitem);
— `[`⟨*i*⟩`,`⟨*j*⟩`,`⟨*k*⟩`]` means the item number ⟨*k*⟩ of the list mentioned in the previous point (a sub-subitem);
— etc.

As in the case of a non-nested list, the index may be negative.
To read items, the syntax of `\readlist` is exactly the same as that for simple (non-nested) lists:

```
\setsepchar{\\/,/ }
\readlist\baz{1,2 a b,3 c\\4 d e f,5,6\\7,,8, ,9 xy z}
a) \string\baz[1] is \baz[1]\par
b) \string\baz[1,1] is \baz[1,1]\par
c) \string\baz[1,1,1] is \baz[1,1,1]\par
b) \string\bar[1,2] is \baz[1,2]\par
e) \string\baz[1,2,3] is \baz[1,2,3]\par
f) \string\baz[-2,1,-1] is \baz[-2,1,-1]
```

a) \baz[1] is 1,2 a b,3 c
b) \baz[1,1] is 1
c) \baz[1,1,1] is 1
b) \bar[1,2] is 2 a b
e) \baz[1,2,3] is b
f) \baz[-2,1,-1] is f

The operator "||" This operator may be employed at any level of nesting.

```
\setsepchar[,]{+||-,*||/}
\readlist\numbers{1+2*3-4/5*6}
Term 1:  \numbers[1]\par
Term 2:  \numbers[2] (factors:  \numbers[2,1] and
         \numbers[2,2])\par
Term 3:  \numbers[3] (factors:  \numbers[3,1],
         \numbers[3,2] and \numbers[3,3])
```

Term 1: 1
Term 2: 2*3 (factors: 2 and 3)
Term 3: 4/5*6 (factors: 4, 5 and 6)

Number of list items The macro `\listlen`⟨*macro-list*⟩`[`⟨*index*⟩`]` requires 2 expansions in order to give the number of items in the list specified by the ⟨*index*⟩. The ⟨*depth*⟩ of the ⟨*index*⟩ must be strictly less than that of the list.
For the case where the ⟨*index*⟩ is empty, `\listlen`⟨*macro-list*⟩`[]`, with 2 expansions, yields the identical result as ⟨*macro-list*⟩`len` with 1 expansion.

```
\setsepchar{\\/,/ }
\readlist\baz{1,2 a b,3 c\\4 d e f,5,6\\7,,8, ,9 xy z}
a) \bazlen\ or \listlen\baz[]\par
b) \listlen\baz[1]\par
c) \listlen\baz[2]\par
d) \listlen\baz[3]\par
e) \listlen\baz[3,1]\par
f) \listlen\baz[3,4]\par% 2 empty items
g) \listlen\baz[3,5]
```

a) 3 or 3
b) 3
c) 3
d) 5
e) 1
f) 2
g) 3

Displaying list items The macro `\showitems`⟨*macrolist*⟩`[`⟨*index*⟩`]` displays items from the list specified by ⟨*index*⟩, in the same manner as `\listlen`. The ⟨*depth*⟩ of the ⟨*index*⟩ must be strictly less than that of the ⟨*list*⟩.

```
\setsepchar{\\/,/ }
\readlist\baz{1,2 a b,3 c\\4 d e f,5,6\\7,,8, ,9 xy z}
a) \showitems\baz[]\par
b) \showitems\baz[1]\par
c) \showitems\baz[2]\par
d) \showitems\baz[3]\par
e) \showitems\baz[3,1]\par
f) \showitems\baz[3,4]\par% 2 empty items
g) \showitems\baz[3,5]
```

a) [1,2 a b,3 c] [4 d e f,5,6] [7„8, ,9 xy z]
b) [1] [2 a b] [3 c]
c) [4 d e f] [5] [6]
d) [7] [8] [] [9 xy z]
e) [7]
f) [] []
g) [9] [xy] [z]

Empty items and extreme (leading/trailing) spaces The removal of empty items and/or leading/trailing spaces will occur in *all* the items, regardless of the degree of nesting. It is clear that a space, "␣", is useless as a parsing separator if you want to use `\readlist*`. Therefore, in the following example, "`*`" is instead selected as the (3rd-tier) parsing separator.
Further, we remove only the extreme spaces, but retain empty items.

```
\setsepchar{\\/,/*}
\readlist*\baz{1, 2*a*b ,3*c\\4*d*e*f,5,6\\7,,8, ,9* xy *z}
a) \showitems\baz[]\par
b) \showitems\baz[1]\par
c) \showitems\baz[2]\par
d) \showitems\baz[3]\par
e) \showitems\baz[3,1]\par
f) \showitems\baz[3,4]\par
g) \showitems\baz[3,5]% "xy" without extreme spaces
```

a) [1, 2*a*b ,3*c] [4*d*e*f,5,6] [7„8, ,9* xy *z]
b) [1] [2*a*b] [3*c]
c) [4*d*e*f] [5] [6]
d) [7] [] [8] [9* xy *z]
e) [7]
f) []
g) [9] [xy] [z]

Iterate over a list The syntax `\foreachitem` ⟨*variable*⟩ `\in` ⟨*macro*⟩`[`⟨*index*⟩`]{`⟨*code*⟩`}` remains valid where now the ⟨*index*⟩ specifies the item (understood as a list) on which to iterate. The ⟨*depth*⟩ of the ⟨*index*⟩ must be strictly less than that of the ⟨*list*⟩.

Assign an item to a macro The syntax `\itemtomacro`⟨*macro-list*⟩`[`⟨*index*⟩`]`⟨*macro*⟩ remains valid to assign to ⟨*macro*⟩ the item specified by ⟨*macro-list*⟩`[`⟨*index*⟩`]`.

```
\setsepchar[,]{\\, }
\readlist\poem{There once was a runner named Dwight\\%
Who could speed even faster than light.\\%
He set out one day\\%
In a relative way\\%
And returned on the previous night.}
\itemtomacro\poem[2]\verse
2nd verse = \verse

\itemtomacro\poem[2,-4]\word
A word = \word
```

2nd verse = Who could speed even faster than light.
A word = even

The macro `\gitemtomacro` makes a global assignment.