

The `coolist` package*

nsetzer

August 14, 2006

The `coolist` package is a “sub” package of the `cool` package that seemed appropriate to publish independently since it may occur that one wishes to include the ability to manipulate lists without having to accept all the overhead of the `cool` package itself.

1 Basics

Lists are defined as a sequence of tokens separated by a comma. The `coolist` package allows the user to access certain elements of the list while neglecting others—essentially turning lists into a sort of array.

Lists elements are accessed by specifying the position of the object within the list (the index of the item) Lists start indexing at 1.

2 Implementation

This is just an internal counter for dealing with the lists, most often used for the length of the list.

```
1 \newcounter{COOL@listlen}%
```

`\setlistEnd` `\setlistStop{<string>}` and `\setlistEnd{<string>}` allow the user to set the end of a list ‘character’s in the rare event that the default values actually appear in the list. Both of these entities are required to properly delimitate the list and avoid errors when macros are included in the list. The default values are

```
2 \newcommand{\COOL@listEnd}{@@@}%
```

```
3 \newcommand{\COOL@listStop}{@@}%
```

and they may be changed by the following commands (which utilize the `\renewcommand`):

```
4 \newcommand{\setlistStop}[1]{\renewcommand{\COOL@listStop}{#1}}%
```

```
5 \newcommand{\setlistEnd}[1]{\renewcommand{\COOL@listEnd}{#1}}%
```

*This document corresponds to `coolist` v1, dated 2006/08/14.

This area defines the core technology behind the `coolist` package: the list “gobbler”. To properly eat a list a counter and a boolean need to be used. `listpointer` acts just like the name implies, as the current “position” of the list. `found` indicates that the position has been found

```
6 \newcounter{COOL@listpointer}%
7 \newboolean{COOL@found}%
```

Now we come to “the gobbler”—a recursive function that eats up a list and gives back the appropriate item. This must be done in \TeX primitives.

The idea behind this is that “the gobbler” eats up everything before the desired item and everything after the desired item.

```
8 \def\COOL@listgobble[#1]#2,#3,\COOL@listEnd{%
9 \ifthenelse{\equal{#3}{\COOL@listStop}}{%
10 {%
```

we have reached the end of the list, just need to check if we need to output something

```
11 \ifthenelse{\value{COOL@listpointer}=#1}%
12 {%
13 \setboolean{COOL@found}{true}%
14 #2%
15 }%
16 % Else
17 {%
18 }%
19 }%
20 % Else
21 {%
22 \ifthenelse{\value{COOL@listpointer}=#1}%
23 {%
24 \setboolean{COOL@found}{true}%
25 #2%
26 }%
27 % Else
28 {%
29 }%
30 \stepcounter{COOL@listpointer}%
```

We must eat up the whole list no matter what or else the stuff beyond `#1` will be displayed. so we need to call “the gobbler” again.

```
31 \COOL@listgobble[#1]#3,\COOL@listEnd%
32 }%
33 }%
```

```
\listval \listval{\langle comma\_delimited\_list \rangle}{\langle index \rangle}
          gives the  $\langle index \rangle$  value of  $\langle comma\_delimited\_list \rangle$ —as in
          \listval{1,2,3,4,5,6}{3} = 3
          $\listval{\alpha,\beta,\gamma}{2}$ =  $\beta$ 
34 \newcommand{\listval}[2]{%
```

start at the beginning of the list, so initialize `listpointer`

```
35 \setcounter{COOL@listpointer}{1}%
```

Assume that the target will not be found—it will be set to true by “the gobbler” if it is

```
36 \setboolean{COOL@found}{false}%
```

Now call the gobbler—since the user shouldn’t be forced to submit the end character (in fact he or she shouldn’t even need to worry that an end character exists nor what it is), we add it on along with the ‘optional’ parameter that tells us which element to retrieve. To ensure that the entire list is read in by `\COOL@listgobbler` we need the list stop ‘character’ too.

```
37 \COOL@listgobble[#2]#1,\COOL@listStop,\COOL@listEnd%
38 }%
```

`\liststore` The list may be stored in a macro of the user’s choosing with the function. The
`\COOL@liststore@gobbler` syntax is

```
\liststore{<csv_list>}{<macro_base_name>}
and the resulting list elements are stored in
<macro_base_name><list_index_roman>
where <list_index_roman> is the list index in roman numerals.
Some examples will clarify:
```

```
\liststore{1,2,3,4}{temp}
\tempi;\tempii;\tempiii;\tempiv yields 1;2;3;4
\liststore{a_1,a_2,a_3,a_4}{temp}
\tempi;\tempii;\tempiii;\tempiv yields a1;a2;a3;a4
39 \def\COOL@liststore@gobbler[#1]#2,#3,\COOL@listEnd{%
40 \ifthenelse{\equal{#3}{\COOL@listStop}}{%
41 {%
42 \expandafter\gdef\csname #1\roman{COOL@listpointer}\endcsname{#2}%
43 }%
44 % Else
45 {%
46 \expandafter\gdef\csname #1\roman{COOL@listpointer}\endcsname{#2}%
47 \stepcounter{COOL@listpointer}%
48 \COOL@liststore@gobbler[#1]#3,\COOL@listEnd%
49 }%
50 }
51 \newcommand{\liststore}[2]{%
52 \setcounter{COOL@listpointer}{1}%
53 \COOL@liststore@gobbler[#2]#1,\COOL@listStop,\COOL@listEnd%
54 }%
```

`\listlen` This returns the length of the list, though it is not useful for storing this length. If you need to record the list’s length for later use, it is better to use the next function `\listlenstore`.

The format is `\listlen{<comma delimited list>}`. It works by recording the value of `listpointer` after it has complete traversed the list. Since indexing

starts at 1, it uses the index 0 which will never ever be an index of the list, so “the gobbler” will not return any value.

Example: `\listlen{1,2,3,4,5} = 5`

```
55 \newcommand{\listlen}[1]{%
56 \listval{#1}{0}%
57 \arabic{COOL@listpointer}
58 }%listlength
```

`\listlenstore` This store the length of the list. The format is `\listlenstore{<counter>}{<comma delimited list>}`.

```
59 \newcommand{\listlenstore}[2]{%
60 \listval{#2}{0}%
61 \setcounter{#1}{\value{COOL@listpointer}}
62 }%listlength
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

C		
<code>\COOL@listEnd</code>	2, 5, 8, 31, 37, 39, 48, 53	<code>\listlenstore</code> <u>59</u>
<code>\COOL@listgobble</code> 8, 31, 37	<code>\liststore</code> <u>39</u>
<code>\COOL@listStop</code> 3, 4, 9, 37, 40, 53	<code>\listval</code> <u>34</u>
<code>\COOL@liststore@gobbler</code> <u>39</u>	
		S
L		
<code>\listlen</code> <u>55</u>	<code>\setlistEnd</code> <u>2</u>
		<code>\setlistStop</code> <u>2</u>