The listliketab package*

Scott Pakin
pakin@uiuc.edu

2000/09/08

Abstract

The `listliketab` package helps the user make list-like `tabular`s, i.e., a `tabular` that is indistinguishable from an `itemize` or `enumerate` environment. The advantage of using a `tabular` is that the user can add additional columns to each entry in the list.

1 Introduction

Here's an itemized list:

- Fee
- Fi
- Fo
- Fum

Here's another itemized list:

- Fee
- Fi
- Fo
- Fum

What's the difference? The two look identical, but the first was typeset in the ordinary way, with an `itemize` environment. The second was typeset within a `tabular` environment, using the `listliketab` package. Because the second is a `tabular`, it can contain additional columns on each line:

*This file has version number v1.00, last revised on 2000/09/08.

• Fee	We can have additional text (and rules) with each item.
• Fi	Try doing *that* in an `itemize` environment!
• Fo	
• Fum	Not so easy, is it?

`listliketab` works with enumerated lists, too:

	Due date
1. Clean desk.	2000/09/10
2. Sort "in" pile.	2000/09/11
3. Discard random applications from "in" pile.	2000/09/11
4. Move applications from "in" pile to "out" pile, stamping each one with a different official-looking stamp.	2000/09/15
5. Write and mail meaningless memos.	2000/09/16
6. Update résumé.	2000/09/20

2 Usage

There are two steps involved in making list-like `tabulars`: First, you store a list environment's parameters. And second, you create a `tabular` using the stored parameters. The following are the commands and environments needed to perform these operations.

`\storestyleof {`⟨*environment*⟩`}`

`\storestyleof` is the easier to use of the two commands that store a list's formatting style. Merely pass this command the name of an existing list environment—generally either `itemize` or `enumerate`—as its ⟨*environment*⟩ parameter. `\storestyleof` will then remember the formatting of that list environment for later use in a `tabular`.

`\storeliststyle`

Sometimes, you have a list environment that takes parameters. `\storestyleof` has no mechanism for passing parameters to such an environment. In this situation, you can manually create a list of the appropriate type and call `\storeliststyle` from within that list. For example:

```
\begin{mylistenvironment}{something}{something else}
  \item[] \storeliststyle{}
\end{mylistenvironment}
```

Note that the above will probably leave some blank space in your document. `\storestyleof`—which uses `\storeliststyle`, incidentally—gets around that problem by building the list within a `minipage` within an `lrbox`. As you can tell, `\storestyleof` is a lot more convenient to use, when applicable.

```
\begin{listliketab}
⟨tabular⟩
\end{listliketab}
```

Once you've stored a list environment's style with `\storestyleof` or `\storeliststyle`, you're ready to typeset list-like `tablular`s. The `listliketab` environment adjusts the internal and external spacing of a `tablular` to match that of the previously given list. It also defines two new field types: `L` and `R`. `L` inserts spacing corresponding to the list's left margin, a right-justified parbox of the same size as the list's label field, and spacing to separate the label from the remaining fields. `R` inserts spacing corresponding to the list's right margin, and is more useful in `tabularx` environments than in ordinary `tabular`s.

Speaking of which, the ⟨*tabular*⟩ you put inside `listliketab` can be any environment that's compatible with the `array` package. This includes `tabular`, `tabularx`, `longtable`, and probably others, as well. Basically, `listliketab` needs to call `\newcolumntype` to define the `L` and `R` fields.

The styles stored by `\storestyleof` and `\storeliststyle` are valid until the next call to one of those commands. Hence, any number of `listliketab` environments can follow a single `\storestyleof` or `\storeliststyle`.

3 Examples

Here's a simple bullet list:

```
\storestyleof{itemize}
\begin{listliketab}
\begin{tabular}{Ll}
  \textbullet & One \\
  \textbullet & Two \\
  \textbullet & Three \\
\end{tabular}
\end{listliketab}
```

and its output:

- One
- Two
- Three

Here's an enumerated list that contains multiple columns after the label:

```
\storestyleof{enumerate}
\begin{listliketab}
  \newcounter{tabenum}\setcounter{tabenum}{0}
  \newcommand{\nextnum}{\addtocounter{tabenum}{1}\thetabenum.}
  \begin{tabular}{L>{\bf}l@{~or~}>{\bf}l@{~or~}>{\bf}l}
    \nextnum & Red   & green & blue \\
    \nextnum & Short & stout & tall \\
    \nextnum & Happy & sad   & confused \\
  \end{tabular}
\end{listliketab}
```

and what it produces:

1. **Red** or **green** or **blue**
2. **Short** or **stout** or **tall**
3. **Happy** or **sad** or **confused**

And finally, here's an example using `tabularx`:

```
\storestyleof{itemize}
\begin{listliketab}
  \begin{tabularx}{0.5\linewidth}{%
      LX@{\raisebox{-2pt}{\framebox(12,12){}}}R}
    \textbullet & Milk \\
    \textbullet & Flour \\
    \textbullet & Sugar \\
    \textbullet & Butter \\
    \textbullet & Eggs \\
  \end{tabularx}
\end{listliketab}
```

and the generated list:

- Milk ☐
- Flour ☐
- Sugar ☐
- Butter ☐
- Eggs ☐

4 Implementation

This section contains the complete source code for `listliketab`. Most users will not get much out of it, but it should be of use to those who need more precise documentation and those who want to extend the `listliketab` package.

```
1 ⟨*package⟩
```

4.1 Utility macros

`\remove@dim` Remove "`pt`" from the end of a dimen (e.g., `12.34pt` ↦ "`12.34`"). I stole this from Hideki Isozaki's `ecltree` package.

```
2 {\catcode`\p=12 \catcode`\t=12 \gdef\remove@dim#1pt{#1}}
```

`\no@pt` Make `\remove@dim` a little more user-friendly.

```
3 \def\no@pt#1{\expandafter\remove@dim\the#1}
```

4.2 List parameter storage

Once we're inside a list environment, we'll need some (global) locations in which to store the local values of various list parameters.

`\llt@labelwidth` `\llt@labelsep` `\llt@topsep` `\llt@rightmargin` These are the global equivalents of `\labelwidth`, `\labelsep`, `\topsep`, and `\rightmargin`, respectively. They're stored by the `\storeliststyle` command from within a list environment.

```
4 \newlength{\llt@labelwidth}
5 \newlength{\llt@labelsep}
6 \newlength{\llt@topsep}
7 \newlength{\llt@rightmargin}
```

`\llt@tab@indent` `\llt@tab@indent` is the indentation of the entire list. It corresponds to the space to the left of the label or, more precisely, `\leftmargin-\labelsep-\labelwidth`.

```
8 \newlength{\llt@tab@indent}
```

`\llt@bot@sep` `\llt@bot@sep` is the amount of space to add at the end of a list. It is set to `\itemsep+\parsep` by `\storeliststyle`. (I'm not actually positive this is the right amount of space to add, but it looks okay to me.)

```
9 \newlength{\llt@bot@sep}
```

4.3 Other variables

`\llt@arraystretch` `\llt@arraystretch@clean` `\llt@arraystretch` is the value we need to assign to `\arraystretch` to make `tabular` spacing mimic the spacing used in the given list. `\llt@arraystretch@clean` is the same as `\llt@arraystretch`, except it is ordinary text instead of a length and does not end in "`pt`".

```
10 \newlength{\llt@arraystretch}
11 \def\llt@arraystretch@clean{}
```

\llt@list@box This box is used by \storestyleof to hold a throwaway list.

```
\newsavebox{\llt@list@box}
```

4.4 Commands

\storeliststyle When \storeliststyle is invoked within a list environment, it does two things. First, it copies the current settings of various list parameters into global variables, so they can be used outside the list. And second, it calculates a value for \arraystretch to match the list's inter-item spacing.

```
\DeclareRobustCommand{\storeliststyle}{
```

Storing list parameters is fairly straightforward.

```
  \setlength{\llt@tab@indent}{\leftmargin-\labelsep-\labelwidth}
  \global\llt@tab@indent=\llt@tab@indent
  \setlength{\llt@bot@sep}{\itemsep+\parsep}
  \global\llt@bot@sep=\llt@bot@sep

  \global\llt@labelwidth=\labelwidth
  \global\llt@labelsep=\labelsep
  \global\llt@rightmargin=\rightmargin
  \global\llt@topsep=\topsep
```

Determining an appropriate value for \arraystretch takes a bit of explanation. Rows of a tabular environment normally have the same height and depth as a strut. Entries in a list are also one strut high/deep, but are separated by \itemsep+\parsep's worth of glue. Hence, to get the new value of \arraystretch, we have to take:

$$\begin{aligned}
\texttt{\textbackslash arraystretch} &= \frac{\text{total space between baselines in a list}}{\text{total space between baselines in a } \texttt{tabular}} \\
&= \frac{\text{item height} + \text{item depth} + \text{inter-item spacing}}{\text{row height} + \text{row depth}} \\
&= \frac{\text{height(strut)} + \text{depth(strut)} + \texttt{\textbackslash itemsep} + \texttt{\textbackslash parsep}}{\text{height(strut)} + \text{depth(strut)}}
\end{aligned}$$

```
  \setlength{\llt@arraystretch}{%
    1.0pt*\ratio{\ht\strutbox+\dp\strutbox+\itemsep+\parsep}
                {\ht\strutbox+\dp\strutbox}}
```

\arraystretch takes a unitless fixed-point number as an argument. Unfortunately, TEX doesn't support such a thing. So we use our \no@pt macro (defined in Section 4.1) to convert from a length to the equivalent text, dropping the units in the process.

```
  \xdef\llt@arraystretch@clean{\no@pt{\llt@arraystretch}}%
}
```

\storestyleof The problem with \storeliststyle is that it can be called only from within a list. What if you don't have a list to use as a template? Well, you have to make

one. Unfortunately, that list then winds up in your document. `\storestyleof` to the rescue! This convenience function creates a list of type `#1` (probably either `itemize` or `enumerate`) containing a call to `\storeliststyle`, but then discards the list environment, so you never see it. (More accurately, `\storelist` constructs the list within an `lrbox` that it never typesets.)

```
\DeclareRobustCommand{\storestyleof}[1]{%
  \begin{lrbox}{\llt@list@box}
  \noindent%
  \begin{minipage}{\linewidth}
    \begin{#1}
      \item[] \storeliststyle{}
    \end{#1}
  \end{minipage}
\end{lrbox}\ignorespacesafterend
}
```

`listliketab` The `listliketab` environment defines a new `tabular` column type, `L`, which corresponds to the list's indentation, the label (a right justified parbox), and the separation between the label and the list body. `L` should be the first field in the user's `tabular` environment. Similarly, listliketab defines `R`, which is the spacing on the right side of the list. `R` is useful when the user is using `tabularx` instead of `tabular`. In that case, a good `tabularx` format string is "`LXR`", possibly with other fields between the `X` and the `R`.

`listliketab` also stretches the array appropriately and suppresses paragraph indentation. (The `L` field will ensure the `tabular` is properly indented.)

```
\newenvironment{listliketab}{%
  \newcolumntype{L}{%
    @{\hspace*{\llt@tab@indent}}%
    >{\hfill}p{\llt@labelwidth}%
    @{\hspace*{\llt@labelsep}}}%
  \newcolumntype{R}{%
    @{\hspace*{\llt@rightmargin}}}%
  \renewcommand{\arraystretch}{\llt@arraystretch@clean}%
  \vspace{\llt@topsep}%
  \noindent\ignorespaces%
}{%
  \vspace{\llt@bot@sep}%
}
```

⟨/package⟩

5 Future work

The `listliketab` environment is too inflexible in terms of defining the `L` and `R` column types for the user's `tabular` environments. First, the user should be able to choose what letters to use, in case he has already assigned a meaning to `L` or `R`. Second, `L` always formats the label as a right-justified parbox, while there may be a case in which the user wants the label to be formatted differently.

The next limitation that should be addressed in a later version of `listliketab` is that the user must manually insert `\textbullet`s (when mimicking `itemize`) or numbers (when mimicking `enumerate`) into the "label" field of his `tabular`. It would be nice if the `listliketab` environment could do this automatically.

Finally, there is no support for nested lists. Those would probably be tricky to mimic properly in a `tabular`, but could occasionally be useful to have.