

The tabularx package

David Carlisle

1999/01/07

Abstract

A new environment,

`\newcolumnntype` These preamble specifications may of course be saved using the command, `\newcolumnntype`, defined in `array.sty`. Thus we may say `\newcolumnntype{Y}{>{\small\raggedright\arraybackslash}X}` and then use `Y` in the `tabularx` preamble argument.

`\tabularxcolumn` The `X` columns are set using the `p` column which corresponds to `\parbox[t]`. You may want them set using, say, the `m` column, which corresponds to `\parbox[c]`. It is not possible to change the column type using the `>` syntax, so another system is provided. `\tabularxcolumn` should be defined to be a macro with one argument, which expands to the `tabular`

```
6 \newdimen\TX@col@width
7 \newdimen\TX@old@table
8 \newdimen\TX@old@col
9 \newdimen\TX@target
10 \newdimen\TX@delta
11 \newcount\TX@cols
12 \newif\ifTX@
```

Now a trick to get the body of an environment into a token register, without doing any expansion. This does not do any real checking of nested environments, so if you should need to nest one tabularx

`\TX@get@body` Place all tokens as far as the first `\end` into a token register. Then call `\TX@find@end` to see if we are at `\end{tabular}`.

```
20 \long\def\TX@get@body#1\end
21   {\toks@ \expandafter{\the\toks@#1}\TX@find@end}
```

`\TX@find@end` If we are at `\end{tabular}`, call `\TX@endtabular`, otherwise add `\end{...}` to the register, and call `\TX@get@body` again.

```
22 \def\TX@find@end#1{%
23   \def\@tempa{#1}%
24   \if\@tempa\TX@ \expandafter\TX@endtabular
25   \else\toks@ \expandafter
26     {\the\toks@ \end{#1}} \expandafter\TX@get@body\fi }
```

`\TX@` The string `tabular` as a macro for testing with `\ifx`.

```
27 \def\TX@{tabular}
```

```

33 \let\@elt\relax
34 \TX@old@table\maxdimen
35 \TX@col@width\TX@target
36 \global\TX@col@s\@ne

```

Typeout some headings (unless this is disabled).

```

37 \TX@typeout@
38 {\@spaces Table Width\@spaces Column Width\@spaces X Columns}%

```

First attempt. Modify the X definition to count X columns.

```

39 \TX@trial{\def\NC@rewrite@X{%
40 \global\advance\TX@col@s\@ne\NC@find p{\TX@col@width}}}%

```

Repeatedly decrease column width until table is the correct width, or stops shrinking, or the columns become too narrow. If there are no multicolumn entries, this will only take one attempt.

```

41 \loop
42 \TX@arith
43 \ifTX@
44 \TX@trial{}%
45 \repeat

```

On last time, with w .316back seneapypndixn

```
56 \TX@col@width\TX@old@col
57 \TX@typeout@{Reached minimum width, backing up.}%
58 \else
```

Otherwise calculate the amount by which the current table is too wide.

```
59 \dimen@wd\@tempboxa
60 \advance\dimen@ -\TX@target
61 \ifdim\dimen@<\TX@delta
```

If this amount is less than \TX@delta , stop. (\TX@delta should be non-zero otherwise we may miss the target due to rounding error.)

```
62 \TX@typeout@{Reached target.}%
63 \else
```

Reduce the number of effective X columns by one. (Checking that we do not get 0, as this would produce an error later.) Then divide excess width by the number

Initialise the X column. The definition can be empty here, as it is set for each tabularx environment.

```
83 \newcolumnntype{X}{}
```

`\tabularxcolumn` The default definition of X is p{#1}.

```
84 \def\tabularxcolumn#1{p{#1}}
```

`\TX@newcol` A little macro just used to cut down the number of `\expandafter` commands needed.

```
85 \def\TX@newcol {\newcol @X}[0]}
```

`\TX@trial`

1. Spaces in the argument are not read verbatim, but may be skipped according to T_EX's usual rules.
2. Spaces will be added to the output after control words, eedSa956.903if-281(the)y-281(w)28(oe)-25

