The tabstackengine Package

Front-end to the stackengine package, allowing tabbed stacking

Steven B. Segletes
steven.b.segletes.civ@mail.mil

November 30, 2016
V2.01

Contents

1 Introduction

The tabstackengine package provides a front end to the stackengine package that allows for the use of tabbing characters within the stacking arguments. **Familiarity with the syntax of the stackengine package is assumed.** When invoked, tabstackengine loads the stackengine package and initializes it in such a way that the end-of-line (EOL) character in certain stacking arguments will be taken, by default, as \\, rather than a space (which is the default EOL

separator in stackengine). The EOL separator can be changed using stackengine's \setstackEOL macro.

With tabstackengine, command variations are introduced to allow several variants of tabbing within the macro arguments. The default tabbing character is the ampersand (&); however, the tabbing character can be reset to other tokens using the \setstackTAB macro.

In most cases (where it makes sense), a stackengine macro name may be prepended with the word tabbed, align, or tabular to create a new tabstackengine macro that allows for tabbed arguments.

2 Modes and Styles of tabstackengine

\TABstackText
\TABstackMath

Like the stackengine package which provides the modes \stackText and \stackMath, the tabstackengine package provides the modes \TABstackText and \TABstackMath. However, the tabstackengine package honors the underlying mode of stackengine, and so if either \stackMath or \TABstackMath are set, all TABstacking arguments will be processed in math mode.[1] So what are the differences?

- \TABstackMath and \TABstackText are local settings, whereas \stackMath and \stackText are global settings.
- As of version 2.00, tabstackengine provides the means to add additional styles to a stack, associated with \TABstackMath and \TABstackText. In particular, the macros \TABstackMathstyle and \TABstackTextstyle can be used to add custom styles to stacks. For example, \TABstackMathstyle{\displaystyle} will cause all stacked items processed in TABstack-math mode to be set in display style. Likewise \TABstackTextstyle{\scriptsize} will cause all stacked items processed in TABstack-text mode to be set in script size. Styles (for both math and text modes) can be cleared with the command \clearTABstyle.

\TABstackMathstyle
\TABstackTextstyle
\clearTABstyle

3 Tabbing Variations within tabstackengine

The tabstackengine package syntax allows three types of tabbing variation denoted by the words tabbed, align, and tabular in the macro name itself. In the case of tabbed macros, the tabbed columns all share the same alignment,

[1] The one exception here is if stackengine macros are embedded (nested) inside tabstackengine macro arguments. In this case, the embedded stackengine macro will only respond to the stackengine mode, and not the tabstackengine mode.

as dictated by the \stackalignment setting or perhaps provided as an optional argument in some macro forms.

In the case of align macros, the alignment in columns is alternately specified as right, then left, *etc.*, in the manner of the align environment of the amsmath package.

Finally, in the case of tabular macros, an extra argument is passed to the macro that specifies the left-center-right alignment for each individual column, in the manner of {lccr}.

4 Column Spacing within tabstackengine

\fixTABwidth

Intercolumn space can be introduced to tabstackengine output in one of two ways. First, there is a macro setting to force all columns to be the same width (namely, the width of the widest entry in the stack), using the syntax \fixTABwidth{T *or* F}. The default is F. When set true, additional column space will be introduced to all but the widest column of a stack, so as to make all columns of a width equal to that of the widest column.

\setstacktabbedgap

Secondly, each of the tabbing variations has the means to introduce a fixed amount of space between columns. By default, the tabbed stacking macros add no space (0pt) between adjacent columns, but this value can be reset with the macro \setstacktabbedgap{*length*}.

\setstackaligngap

In the case of the align stacking macros, there is never any gap introduced after the right-aligned (odd-numbered) columns. However, the default gap introduced after the left-aligned (even-numbered) columns is, by default, 1em (the same gap as \quad). It can be reset with the macro \setstackaligngap{*length*}.

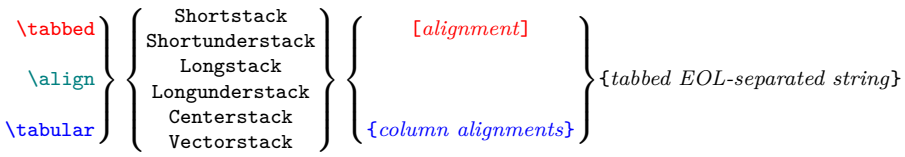
\setstacktabulargap

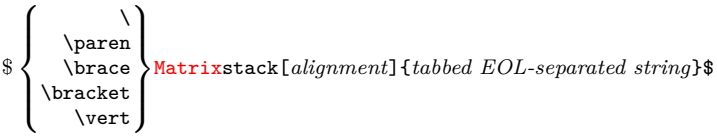
For the tabular stacks, the default intercolumn gap is the value of \tabcolsep. The default value may be reset with the macro \setstacktabulargap{*length*}.

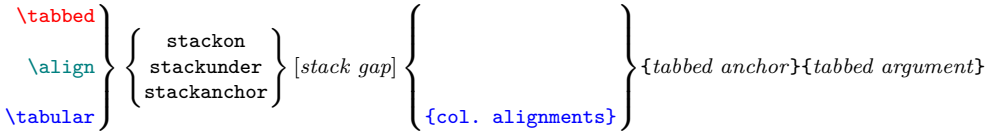
Note that these \setstack...gap macros are for setting horizontal gaps between columns of a stack. They should not be confused with the \setstackgap macro of stackengine that sets the vertical gap for long and short stacks.

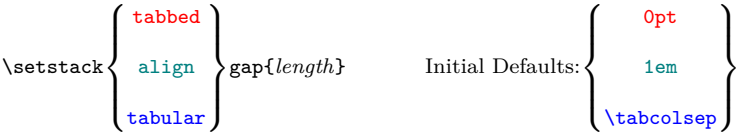
5 Command Summary

Below are the new TABstack making commands introduced by tabstackengine. In the syntax shown below, when there are multiple commands delimited by braces, any one of the commands within the brace may be selected.









`\ensureTABstackMath` The macro

`\ensureTABstackMath{`*commands involving TABstacks*`}`

will force any tabstackengine stacks within its argument to be processed in math mode, even if the prevailing mode is otherwise `\TABstackText`. The package also provides a set of declarations that can be used to define the manner in which subsequent TABstacks will be processed:

`\fixTABwidth{T` *or* `F}`
`\TABstackMath`
`\TABstackText`
`\TABstackMathstyle{`*directive*`}`
`\TABstackTextstyle{`*directive*`}`
`\clearTABstyle`
`\setstackEOL{`*end-of-line character*`}` (provided by stackengine)
`\setstackTAB{`*tabbing character*`}`
`\TABunaryLeft` (`\TABbinaryRight`)
`\TABunaryRight` (`\TABbinaryLeft`)
`\TABbinary`

The following macros can be used for parsing tabbed data outside of a TABstack and also provide various stack metrics for the most recently parsed tabstackengine data.

```
\readTABstack{tabbed EOL-separated string}
\TABcellRaw[row,column]
\TABcell{row}{column}
\TABcellBox[alignment]{row}{column}
\getTABcelltoks[row,column]     \the\TABcelltoks
\TABcells{row or blank}
\TABstrut{row}
\TABwd{column}
\TABht{row}
\TABdp{row}
```

5.1 Command Examples

Below we give examples of the various types of commands made available through the tabstackengine package.

Tabbed End-of-Line (EOL)-delimited Stacks

Here, the optional argument `[l]` defines the alignment of *all* the columns as "left." The default alignment is `[c]`.

```
\TABstackTextstyle{\scshape}
\tabbedShortunderstack[l]{This& Is &The\\Time & Of&Man's\\ Great&Dis&content}
```

THIS IS THE
TIME OF MAN'S
GREATDISCONTENT

Note that spaces around the arguments are absorbed and discarded. Furthermore, the text style has been set to `\scshape`.

Align End-of-Line (EOL)-delimited Stacks

In an `align`-stack, the column alignments will always be `rlrl...` The gap following the left-aligned columns is set by `\setstackalignngap`.

```
$\ensurestackMath{Z:\left\{\alignCenterstack{%
  y=&mx+b,&0=&Ax+By+C \\ y_1=&W_1,&y_2=&W_2}\right.}$
```

$$Z:\begin{cases} y = mx + b, & 0 = Ax + By + C \\ y_1 = W_1, & y_2 = W_2 \end{cases}$$

Tabular End-of-Line (EOL)-delimited Stacks

In a `tabular`-stack, the alignment of each column is specified in a separate leading argument.

```
\stackText\tabularLongstack{rllc}{%
  9)& $y_1=mx+b$ &linear&*\\10)& $y_2=e^x$ &exponential&[23]}
```

9) $y_1 = mx + b$ linear *
10) $y_2 = e^x$ exponential [23]

Matrix Stacks

The `Matrix`-stacks are tabbed variants of stackengine's `Vector`-stacks.

```
\setstacktabbedgap{1.5ex}
$I = \bracketMatrixstack{1&0&0\\0&1&0\\0&0&1}$
```

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tabbed Stack

This variant of a `tabbed`-stack stacks exactly two items. The optional argument is a stacking gap, as in the syntax of the stackengine package.

```
\setstacktabbedgap{1ex}
\tabbedstackon[4pt]{Jack&drove&the car&home.}{SN&V&DO&IO}
```

SN V DO IO
Jack drove the car home.

Align Stack

This is for stacking two items with `rlrl...` alignment pattern.

```
\TABstackMath\setstackalignгap{3em}
\alignstackunder[8pt]{y=&mx+b,&0=&Ax+By+C}{y_1=&W_1,&y_2=&W_2}
```

$y = mx + b,$ $0 = Ax + By + C$
$y_1 = W_1,$ $y_2 = W_2$

Tabular Stack

This is for stacking two items with specifiable alignment pattern.

```
\TABbinary\TABstackMath\setstacktabulargap{1ex}%
  \tabularstackanchor[-1pt]{rcl}%
  {\rule{7ex}{1pt}&\belowbaseline[0pt]{$\triangle$}%
    &\rule{7ex}{1pt}}{1 + 3(4-3) &=& 7 - 6/2}
```

$$\begin{array}{c@{}c@{}c} \overline{} & \triangle & \overline{} \\ 1 + 2(4-3) & = & 6 - 6/2 \end{array}$$

Note the use of `\TABbinary`, which applies a group to the beginning and end of each cell, in the event a binary treatment of leading/trailing operators is desired. So, in this case, a cell containing `=` will be set as `{}={}`. In the absence of that declaration, the cell containing the equal sign would have to have been explicitly defined as `{}=` in order to override the `\unaryLeft` default setting of the package, which only places a group at the end of each cell. Note also the use of a negative stacking gap, which is a perfectly acceptable syntax and can be used to achieve overlap, if desired.

Fixed Tab Width (equal width columns, based on largest)

`\fixTABwidth` With the `\fixTABwidth` mode set, the stack will have fixed-width columns, based on the overall widest entry. Compare with versus without fixed width for the following TABstack.

```
\setstacktabbedgap{1ex}\fixTABwidth{T}%
$\left(\tabbedCenterstack[r]{1&34&544\\4324329&0&8\\89&123&1}\right)$
versus \fixTABwidth{F}%
$\left(\tabbedCenterstack[r]{1&34&544\\4324329&0&8\\89&123&1}\right)$
```

$$\begin{pmatrix} 1 & 34 & 544 \\ 4324329 & 0 & 8 \\ 89 & 123 & 1 \end{pmatrix} \text{ versus } \begin{pmatrix} 1 & 34 & 544 \\ 4324329 & 0 & 8 \\ 89 & 123 & 1 \end{pmatrix}$$

Setting the Stack Tabbing Character

By default, for the parsing of columns within a given row, this package employs the `&` character to delimit the columns. This value can be changed via
`\setstackTAB` `\setstackTAB{`*tabbing character*`}`, where the argument is the newly desired tabbing token. It can be any of various tokens, including a space token, if one wishes to parse a space-separated list of columns.

TABstacks Inside the `tabular` Environment

When invoking a TABstack inside another tabbed environment, such as `tabular`, `align`, or other similar environments, one must group the TABstacks in their own braces `{}`:

```
\ensureTABstackMath{%
\begin{tabular}{c|c}
Left Eqn. & Right Eqn.\\
\hline
{\tabularCenterstack{lr}{a_1 & 12\\c & 1234}} & \
{\tabularCenterstack{rl}{a_1 & 12\\c & 1234}}\\
\hline
\end{tabular}
}
```

Left Eqn.	Right Eqn.
a_1 12	a_1 12
c 1234	c 1234

Math Relations/Operators at Cell Left/Right Extrema

There are two things to keep in mind regarding TABstacked content. First, a TABstack cell has no precise understanding up what content precedes it in the cell to the immediate left, nor what content follows it in the cell to the immediate right. It does, however know the overall height/depth of the content across the whole row and creates a vertical "strut" of that height and depth, which must, in some way, be applied to every cell in the row.

This vertical strut can be applied to the cell immediately prior to or immediately following the cell content, as we shall see. However, such an action will have an effect on math operators and relations found at the leading or trailing ends of the cell content.

Math operators and relations can be categorized as unary or binary; some may be both, depending on their usage context, such as the minus sign. When used as $a - b$, the relation is binary, because it connects a and b in a mathematical operation. Note how space appears both before and after the minus sign. Alternatively, when used as $-\pi$, the minus sign operates only upon what follows, in this case π, to produce a negative. Note how no space is introduced between the minus sign and π. This is the minus used as a unary operator.

Because a TABstack cell has no intimate knowledge of the adjacent cell content, it is up to the user to employ his tabbing separators in a way that produces the desired result. By default, tabstackengine will place the strut after the cell content. This means that any trailing math operator in a cell will present itself

in its binary form (regardless of what comes in the cell to the right), because the strut will appear as trailing data against which the operator can be set. Similarly, any leading math operator will present itself as unary (regardless of what content appears in the cell to the left).

Thus, under the default setting `\tabbedLongstack{y =&-mx +& b}` will present as $y = -mx + b$, by default, with the trailing equal and plus signs as binary, and the leading minus sign as unary. The package can reverse the default with the following declarative modes: `\TABunaryRight` (identical to `\TABbinaryLeft`); alternately, one may use `\TABbinary`, which will present both leading *and* trailing operators in their binary form. The default can be restored with `\TABunaryLeft` (identical to `\TABbinaryRight`).

`\TABunaryRight`
`\TABbinaryLeft`
`\TABbinary`
`\TABunaryLeft`
`\TABbinaryRight`

Without changing any of the package strut modes, an operator, such as minus, can always be forced into its unary mode by enclosing it in braces: `{-}`. Likewise, it can be forced into its binary mode by placing empty braces on both sides of it: `{}-{}`.

The Parsing Macros `\readTABstack`, *etc.*

As of version 2.00 of the tabstackengine package, the parsing functions of the package were delegated to the very powerful listofitems package. As such **the `\readTABrow` macro is no longer supported**. For typical parsing functionality, therefore, please consult the documentation to the listofitems package and its `\readlist` macro. I commend it to your inspection and use for a variety of parsing tasks.

However, there may still be a need to access the various stacking related data in either a recently composed TABstack, or even one that is yet to be typeset. When a TABstack is constructed by the tabstackengine package, a call is made to the routine `\readTABstack`, in order to parse the data. This macro may be independently called by the user to read TABstack data without producing a constructed TABstack, by passing it the same tabbed, EOL-separated data that would otherwise be used to construct a stack.[2] If the routine is not called independently by the user, data from the most recent TABstacking operation is still available for interrogation.

`\readTABstack`

Take the example

```
\TABstackMath\TABstackMathstyle{\displaystyle}\setstackgap{S}{5pt}%
\alignShortstack{\frac{A}{Q}x=&B\\ C= &\frac{Dx}{2}\\E=&F}
```

[2] Alternately, TABstacking data can be read without generating a TABstack by using the `\renewcommand\quietstack{T}` setting of stackengine to suppress the stack output, without suppressing its construction.

which presents as

$$\frac{A}{Q}x = B$$
$$C = \frac{Dx}{2}$$
$$E = F$$

\TABwd \TABht \TABdp

Let us say we were interested in information about the cell in the 1st column of the 2nd row. I can obtain its dimensions as the column-1 width `\TABwd{1}`, as well as the row-2 height and depth `\TABht{2}` and `\TABdp{2}`. Note that these macros provide dimensions of the TABstack *cell*, which in this case is larger than the mere "$C =$" content. Those dimensions are as follows, followed by a `\rule` depicting the total size of the cell:

Width: 29.35422pt, Height/Depth: 13.59839pt/6.85951pt, Rule: 

\TABcellRaw

One can also obtain information about what is in the cell. Here, use the macro `\TABcellRaw[2,1]`, which will expand to the tokens employed in the stack definition (shown here in an `\fbox` to show that the leading/trailing spaces have been discarded):

$\boxed{\text{C=}}$

\TABcell

If one would like to see the cell data presented in the prevailing (tab)stackengine mode and style[3], the macro `\TABcell{2}{1}` may be used (again shown in an `\fbox`):

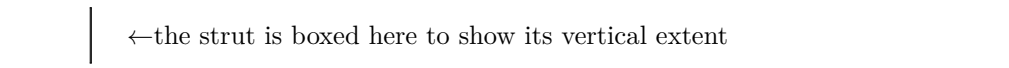
$\boxed{C =}$

Note, however, that the `\TABcell` still does not account for three things:

- it is not strutted to reflect the height of the full row content;
- it does not reflect the full column width (nor the alignment within the column); and
- it does not provide any of the empty group treatments that would otherwise make leading/trailing math operators perform in a binary fashion.

\TABstrut

A strut of the given row height may be obtained with `\TABstrut{2}`:



[3]Note that both `\TABcell` and, as described later, `\TABcellBox` present in the *prevailing* TABstack mode and style. While a recent use of `\ensureTABstackMath` will be remembered, intervening declarations of `\TABstackMath`, `\TABstackText` and their associated styles will change the *prevailing* mode and style in which subsequent `\TABcell` and `\TABcellBox` are processed.

\TABcellBox

However, to obtain the fully rendered cell, *as it appears within the actual TABstack*, one needs `\TABcellBox{2}{1}`, shown (in an `\fbox`) as

$\boxed{C=}$

Since the `\readTABstack` macro, itself, neither knows nor determines the eventual cell alignment of a future stack, the actual `lcr` alignment of a `\TABcellBox` will only be known when applied to a previously constructed stack. Therefore, if `\TABcellBox` is called following an independent invocation of `\readTABstack`, center alignment of the cell content will be provided, by default, which can be overridden with the optional argment to `\TABcellBox`.

Note that the height/depth of the `\TABcellBox` reflects the height and depth of the row content of the TABstack. For short stacks, the specified gap between rows is *in addition* to these strutted boxes. For long stacks, the inter-row spacing is independent of the box height and depth. However, even for long stacks, the height of the top row and the depth of the bottom row of a stack still affect the overall dimensions of the stack.

If one wishes to recover the *actual tokens* that were employed in a given TABstack cell (rather that just something that will *expand* to those tokens), that can be accomplished in one of two ways. The macro `\TABcellRaw[,]` can be expanded twice in the manner of

```
\detokenize\expandafter\expandafter\expandafter{\TABcellRaw[2,2]}
```

```
\frac {Dx}{2}
```

\getTABcelltoks
\the\TABcelltoks

Alternately, the macro `\getTABcelltoks[,]` will produce a token list named `\TABcelltoks` that contains the cell's tokens:

```
\getTABcelltoks[2,2]\detokenize\expandafter{\the\TABcelltoks}
```

```
\frac {Dx}{2}
```

In summary then, tabstackengine cell content can be accessed in a number of ways:

`\TABcellRaw[,]` – expands into the tokens of the cell

`\TABcell{}{}` – presents the cell content in the prevailing mode (text or math) and style set by stackengine and tabstackengine

`\TABcellBox{}{}` – presents the cell content, in the prevailing mode and style, strutted to the proper row height/depth, set in a box of the proper cell width, flanked by the appropriate `{}` groups defined by tabstackengine's unary and/or binary declarations, and (when knowable) set in the proper `lcr` alignment

`\getTABcelltoks[,]` – creates a token list register `\TABcelltoks` that contains the actual tokens employed in the cell, accessible by way of `\the\TABcelltoks`

TABstack Array Dimensions

Consider the example

```
\setstacktabbedgap{.5em}
\tabbedLongstack{a & b & c & d\\ e & f & g & h\\ i & j & k & l}
```

which produces

$a\ b\ c\ d$
$e\ f\ g\ h$
$i\ j\ k\ l$

The macros `\TABwd`, `\TABht`, and `\TABdp` were presented as the means to get the physical dimensions of various rows and columns of a TABstack. But what if the information sought is the number of rows and columns?

`\TABcells` The macro `\TABcells{}` performs the function. When passed a blank argument, it returns the number of rows of the most recently constructed TABstack (or `\readTABstack`).

Rows = `\TABcells{}` = 3

On the other hand, pass it a row number for its arguments and it will tell you how many columns below to that row

Columns = `\TABcells{1}` = 4

Note that tabstackengine uses the number of columns provided in row 1 to determine the dimensions of the subsequent TABstack. If the 2nd or 3rd rows of the above stack were [accidentally] defined with 5 columns of data, the 5th column of data would be ignored during the TABstack construction, since the

1st row only has 4 columns. However, in that case, `\TABcells{2}` would still, in fact, yield 5.

6 Absent Features/Tricky Syntax

1. Nothing Equivalent to `\hline` or `|`

This is not a bug, but rather a notation of a missing feature. Currently there is nothing equivalent to `\hline` available for use in tabstackengine arguments. Furthermore, vertical lines may **not** be added to a tabular stack with the use of `|` elements in the column specifier.

2. Empty Items Are Not Ignored (by Default)

The listofitems package used to parse TABstack input, does not, by default, ignore empty items. This can cause parsing errors, if not understood properly. Take, for example, the well formed TABstack invocation,

`\tabularLongstack{rc}{11&12\\21&22\\31&32}`.

Adding a trailing `\\` to the input, as in:

`\tabularLongstack{rc}{11&12\\21&22\\31&32\\}`,

however, breaks the parsing because 2 columns of data are expected following the final `\\` (even though such syntax is accepted in, for example, the `tabular` environment). This syntax can be immediately made acceptable by invoking the listofitems declaration `\ignoreemptyitems`, in which case the final [empty] row is discarded. However, that approach can introduce a new set of problems, because it will then ignore actual blank input that was intended, as in the case of this example, in which table cell (2,2) is intentionally left blank:

`\tabularLongstack{rc}{11&12\\21&\\31&32\\}`.

To make this latter case work, when empty items are ignored, an empty group would need be explicitly inserted:

`\tabularLongstack{rc}{11&12\\21&{}\\31&32\\}`.

These problems can be wholly avoided if care is used in the construction of TABstack input.

Acknowledgements

I would like to thank Christian Tellechea for his development of the listofitems package (which was directly inspired by my deficient getargs package). The macros provided by Christian were directly implemented for version 2.00 of the tabstackengine package.

I would also thank the user "Werner" at tex.stackexchange.com for helping me to understand some of the details of the etoolbox package:

http://tex.stackexchange.com/questions/140372/
loop-multi-contingency-using-etoolbox

7 Code Listing

```
\def\tabstackengineversionnumber{V2.01}
%
% THIS MATERIAL IS SUBJECT TO THE LaTeX Project Public License
%
% V1.00 -Adopted beta version 0.21 as initial release version 1.0
% V1.10 -Corrected unary/binary problem for left end of tabbed cell content;
%       -Added \TABunaryLeft (\TABbinaryRight) for "   cell{} ";
%        added \TABunaryRight (\TABbinaryLeft) for " {}cell   ";
%        added \TABbinary                      for " {}cell{} ";
%        The default is \TABunaryLeft (V1.00 wrongly equivalent to \TABbinary)
%        This removes need to brace unary negatives at lead of cell.
%       -Corrected bug of trailing \frac, noted in V1.00, by adding a
%        \relax to definition of \@postTAB in \readTABrow.
% V2.00 -Incorporate listofitems package methodology for parsing, requiring
%        some package rewrite, primarily macro \@readMANYrows.
%       -Fixed bug in \ensureTABstackMath that had automatically returned to
%        \TABstackText mode.
%       -Added \Matrixstack macro (equivalent to \tabbedVectorstack)
%       -Added \TABstackTextstyle, \TABstackMathstyle, and \clearTABstyle
%        to allow things like fontsize, \displaystyle, etc. to be set inside
%        a stack, by default.
%       -Converted \newlength\maxTAB@width to a \xdef\maxTABwd.  Introduced
%        \TABwd{}, \TABht{}, and \TABdp{} for obtaining widths of columns
%        and heights/depths of rows.
%       -\TABstrut now defined at time of parsing, rather than reconstructed after
%        the fact.  This will prevent any confusions if the math/text stack mode
%        settings change.
%       -Employed a \toks based approach to parsing the argument, rather than the
%        previous \protected@edef approach.  The prior approach suffered problems
%        when \\ was redefined, as in, for example, \centering or \raggedright.
% V2.01 -Take advantage of listofitems revision to allow global \readlist, via
%        \greadlist.  This allows \setsepchar to be placed inside of group, thus
%        preventing a global change in the listofitems \setsepchar.
%       -\TABcell and \TABcellBox modified to remember recent use of
%        \ensureTABstackMath, which otherwise changes temporarily the
```

```
%        prevailing mode and style of the TABstack.
\ProvidesPackage{tabstackengine}
[2016/11/30 (\tabstackengineversionnumber) tabbed stacking]
\RequirePackage{stackengine}[2016-10-04]
\RequirePackage{listofitems}[2016/11/18]
\RequirePackage{etoolbox}

\newcounter{TABrowindex@}
\newcounter{TABcolindex@}
\newcounter{TABalignmentindex@}
\newtoggle{fixed@TABwidth}
\newtoks\TABcelltoks
\newtoks\TABcoltoks
\newtoks\LstrutTABtoks
\newtoks\RstrutTABtoks

\def\getTABcelltoks[#1,#2]{%
  \TABcelltoks=\expandafter\expandafter\expandafter{\TABcellRaw[#1,#2]}}

\def\@getstruttedTABcelltoks[#1,#2]{%
  \getTABcelltoks[#1,#2]%
  \edef\TABstack@rownum{#1}%\
  \LstrutTABtoks=\expandafter{\expandafter\TAB@strutL\expandafter{\TABstack@rownum}}%
  \RstrutTABtoks=\expandafter{\expandafter\TAB@strutR\expandafter{\TABstack@rownum}}%
  \prepend@toksto\TABcelltoks\LstrutTABtoks%
  \append@toksto\TABcelltoks\RstrutTABtoks}

\def\prepend@toksto#1#2{#1=\expandafter\expandafter\expandafter%
  {\expandafter\the\expandafter#2\the#1}}

\def\append@toksto#1#2{#1=\expandafter\expandafter\expandafter%
  {\expandafter\the\expandafter#1\the#2}}

\newcommand\setstackTAB[1]{\ifstrempty{#1}{\def\TAB@char{ }}{\def\TAB@char{#1}}}

\newcommand\readTABstack[1]{%
  \expandafter\expandafter\expandafter\setsepchar\expandafter\expandafter%
    \expandafter{\expandafter\SEP@char\expandafter/\TAB@char}%
  \greadlist*\TABcellRaw{#1}%
  \edef\TABstack@rows{\TABcellRawlen}%
  \edef\TABstack@cols{\listlen\TABcellRaw[1]}%
  \def\maxTABwd{0pt}%
  \setcounter{TABrowindex@}{0}%
  \whileboolexpr{test {\ifnumless{\theTABrowindex@}{\TABstack@rows}}}{% ROW LOOP
    \def\@accumulatedTAB{}%
    \stepcounter{TABrowindex@}%
    \setcounter{TABcolindex@}{0}%
    \whileboolexpr{test {\ifnumless{\theTABcolindex@}{\TABstack@cols}}}{% COL LOOP
      \stepcounter{TABcolindex@}%
      \ifnum\value{TABrowindex@}=1\relax\csxdef{col\theTABcolindex@ TAB@wd}{0pt}\fi%
      \getTABcelltoks[\theTABrowindex@,\theTABcolindex@]%
      \expandafter\g@addto@macro\expandafter\@accumulatedTAB\expandafter{%
        \the\TABcelltoks{}}%
      \setbox0=\hbox{\stack@delim\TAB@delim{%
        \TAB@strutL{0}\the\TABcelltoks\TAB@strutR{0}}\stack@delim}%
      \ifdim\wd0>\csuse{col\theTABcolindex@ TAB@wd}\relax%
        \csxdef{col\theTABcolindex@ TAB@wd}{\the\wd0}%
```

```
      \ifdim\wd0>\maxTABwd\relax\xdef\maxTABwd{\the\wd0}\fi\fi%
      \csxdef{col\theTABcolindex@ TAB@stackalignment}{c}% DEFAULT, LATER CHANGED
    }%
    \setbox0=\hbox{\stack@delim\TAB@delim{\@accumulatedTAB}\stack@delim}%
    \csxdef{row\theTABrowindex@ TAB@ht}{\the\ht0}%
    \csxdef{row\theTABrowindex@ TAB@dp}{\the\dp0}%
    \global\let\recent@TAB@delim\TAB@delim%
  }%
}

\newcommand\TABwd[1]{\csuse{col#1TAB@wd}}
\newcommand\TABht[1]{\csuse{row#1TAB@ht}}
\newcommand\TABdp[1]{\csuse{row#1TAB@dp}}
\newcommand\TABstrut[1]{\ifnum#1<1\relax{}\else%
  \protect\rule[-\TABdp{#1}]{0pt}{\dimexpr\TABdp{#1}+\TABht{#1}\relax}\fi}

\newcommand\TABcell[2]{\stack@delim\recent@TAB@delim{\TABcellRaw[#1,#2]}\stack@delim}

\newcommand\TABcellBox[3][\relax]{\ifx\relax#1\relax%
  \TABcellBox@aux{\csuse{col#3TAB@stackalignment}}{#2}{#3}\else
  \TABcellBox@aux{#1}{#2}{#3}\fi}

\newcommand\TABcellBox@aux[3]{\makebox[\TABwd{#3}][#1]{\stack@delim%
  \recent@TAB@delim{\TAB@strutL{#2}\TABcellRaw[#2,#3]\TAB@strutR{#2}}\stack@delim}}

\newcommand\TABcells[1]{\listlen\TABcellRaw[#1]}

\newcommand\TABunaryLeft{\def\TAB@strutL##1{}%
                         \def\TAB@strutR##1{\TABstrut{##1}}}
\newcommand\TABunaryRight{\def\TAB@strutL##1{\TABstrut{##1}}%
                          \def\TAB@strutR##1{}}
\newcommand\TABbinary{\def\TAB@strutL##1{{}}%
                      \def\TAB@strutR##1{\TABstrut{##1}}}
\let\TABbinaryRight\TABunaryLeft
\let\TABbinaryLeft\TABunaryRight

\newcommand\TABstackMath{\renewcommand\TAB@delim[1]{\ensuremath{\TAB@mathstyle##1}}%
  \let\recent@TAB@delim\TAB@delim}
\newcommand\TABstackText{\renewcommand\TAB@delim[1]{\TAB@textstyle##1}%
  \let\recent@TAB@delim\TAB@delim}
\newcommand\TABstackMathstyle[1]{\renewcommand\TAB@mathstyle{#1}}
\newcommand\TABstackTextstyle[1]{\renewcommand\TAB@textstyle{#1}}
\newcommand\clearTABstyle{\renewcommand\TAB@textstyle{}\renewcommand\TAB@mathstyle{}}

\newcommand\tabbedShortstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Shortstack}}

\newcommand\alignShortstack[1]{%
  \@TAB@stack{}{#1}{A}{\Shortstack}}

\newcommand\tabularShortstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Shortstack}}

\newcommand\tabbedShortunderstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Shortunderstack}}

\newcommand\alignShortunderstack[1]{%
```

```
  \@TAB@stack{}{#1}{A}{\Shortunderstack}}

\newcommand\tabularShortunderstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Shortunderstack}}

\newcommand\tabbedLongstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Longstack}}

\newcommand\alignLongstack[1]{%
  \@TAB@stack{}{#1}{A}{\Longstack}}

\newcommand\tabularLongstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Longstack}}

\newcommand\tabbedLongunderstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Longunderstack}}

\newcommand\alignLongunderstack[1]{%
  \@TAB@stack{}{#1}{A}{\Longunderstack}}

\newcommand\tabularLongunderstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Longunderstack}}

\newcommand\tabbedCenterstack[2][\stackalignment]{%
  \@TAB@stack{#1}{#2}{D}{\Centerstack}}

\newcommand\alignCenterstack[1]{%
  \@TAB@stack{}{#1}{A}{\Centerstack}}

\newcommand\tabularCenterstack[2]{%
  \@TAB@stack{}{#2}{#1}{\Centerstack}}

\newcommand\tabbedVectorstack[2][\stackalignment]{%
  \ensureTABstackMath{\@TAB@stack{#1}{#2}{D}{\Vectorstack}}}

\newcommand\alignVectorstack[1]{%
  \ensureTABstackMath{\@TAB@stack{}{#1}{A}{\Vectorstack}}}

\newcommand\tabularVectorstack[2]{%
  \ensureTABstackMath{\@TAB@stack{}{#2}{#1}{\Vectorstack}}}

\let\Matrixstack\tabbedVectorstack% ADDED V2.00

\newcommand\parenMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left(\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right)}}

\newcommand\braceMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left\{\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right\}}}

\newcommand\bracketMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left[\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right]}}

\newcommand\vertMatrixstack[2][\stackalignment]{%
  \ensureTABstackMath{\left|\@TAB@stack{#1}{#2}{D}{\Vectorstack}\right|}}

\newcommand\@TAB@stack[4]{\bgroup%
  \readTABstack{#2}%
```

```
  \edef\stackalignment{#1}%
  \setcounter{TABcolindex@}{0}%
  \whileboolexpr{test {\ifnumless{\theTABcolindex@}{\TABstack@cols}}}{% COL LOOP
    \stepcounter{TABcolindex@}%
    \@getstruttedTABcelltoks[1,\theTABcolindex@]%
    \TABcoltoks=\expandafter{\expandafter\TAB@delim\expandafter{\the\TABcelltoks}}%
    \@getTABalignment{#3}{\theTABcolindex@}%
    \ifboolexpr{test {\ifnumgreater{\theTABcolindex@}{1}}}%
      {\add@TAB@gap{#3}{\theTABcolindex@}}{}%
    \setcounter{TABrowindex@}{1}%
    \whileboolexpr{test {\ifnumless{\theTABrowindex@}{\TABstack@rows}}}{% ROW LOOP
      \stepcounter{TABrowindex@}%
      \@getstruttedTABcelltoks[\theTABrowindex@,\theTABcolindex@]%
      \TABcoltoks=\expandafter\expandafter\expandafter\expandafter\expandafter%
        \expandafter\expandafter{\expandafter\expandafter\expandafter%
        \the\expandafter\expandafter\expandafter\TABcoltoks\expandafter%
        \SEP@char\expandafter\TAB@delim\expandafter{\the\TABcelltoks}}%
    }%
    \iftoggle{fixed@TABwidth}%
      {\makebox[\maxTABwd][\stackalignment]{%
        \expandafter#4\expandafter{\the\TABcoltoks}}}%
      {\expandafter#4\expandafter{\the\TABcoltoks}}%
  }%
\egroup}

\newcommand\tabbedstackon[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackon}}

\newcommand\alignstackon[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackon}}

\newcommand\tabularstackon[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackon}}

\newcommand\tabbedstackunder[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackunder}}

\newcommand\alignstackunder[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackunder}}

\newcommand\tabularstackunder[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackunder}}

\newcommand\tabbedstackanchor[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{D}{\stackanchor}}

\newcommand\alignstackanchor[3][\stackgap]{%
  \@TABstackonunder{#1}{#2}{#3}{A}{\stackanchor}}

\newcommand\tabularstackanchor[4][\stackgap]{%
  \@TABstackonunder{#1}{#3}{#4}{#2}{\stackanchor}}

\newcommand\@TABstackonunder[5]{\bgroup%
  \def\TAB@tmp{#2}%
  \expandafter\g@addto@macro\expandafter\TAB@tmp\expandafter{\SEP@char#3}%
  \readTABstack{\TAB@tmp}%
  \setcounter{TABcolindex@}{0}%
```

```
  \whileboolexpr{test {\ifnumless{\theTABcolindex@}{\TABstack@cols}}}{% COL LOOP
    \stepcounter{TABcolindex@}%
    \@getTABalignment{#4}{\theTABcolindex@}%
    \ifboolexpr{test {\ifnumgreater{\theTABcolindex@}{1}}%
    }{\add@TAB@gap{#4}{\theTABcolindex@}}{}%
    \iftoggle{fixed@TABwidth}%
      {\makebox[\maxTABwd][\stackalignment]{%
       #5[#1]%
         {\TAB@delim{\TAB@strutL{1}\TABcellRaw[1,\theTABcolindex@]\TAB@strutR{1}}}%
         {\TAB@delim{\TAB@strutL{2}\TABcellRaw[2,\theTABcolindex@]\TAB@strutR{2}}}}}%
      {#5[#1]%
         {\TAB@delim{\TAB@strutL{1}\TABcellRaw[1,\theTABcolindex@]\TAB@strutR{1}}}%
         {\TAB@delim{\TAB@strutL{1}\TABcellRaw[2,\theTABcolindex@]\TAB@strutR{2}}}}%
  }%
\egroup}

\newcommand\@getTABalignment[2]{%
  \ifstrequal{#1}{D}{}{%                     T, DO NOTHING (USE \stackalignment)
    \ifstrequal{#1}{A}{%
      \ifnumequal{1}{#2}{%
        \def\stackalignment{r}}{%            A, 1st ELEMENT, SET TO r
        \if l\stackalignment%
          \def\stackalignment{r}\else%       A, SWITCH l TO r
          \def\stackalignment{l}\fi}}{%      A, SWITCH r TO l
          \set@tabularcellalignment{#1}{#2}% tabular, READ #2 location
      }%
  }%
  \csxdef{col\theTABcolindex@ TAB@stackalignment}{\stackalignment}%
}

\newcommand\setstacktabbedgap[1]{\def\tabbed@gap{#1}}
\newcommand\setstackalignngap[1]{\def\align@gap{#1}}
\newcommand\setstacktabulargap[1]{\def\tabular@gap{#1}}

\newcommand\add@TAB@gap[2]{%
  \ifstrequal{#1}{D}{\hspace{\tabbed@gap}}{%
    \ifstrequal{#1}{A}%
      {\if r\stackalignment\hspace{\align@gap}\fi}{\hspace{\tabular@gap}}%
  }%
}

\newcommand\set@tabularcellalignment[2]{%
  \setcounter{TABalignmentindex@}{1}%
  \edef\tabular@settings{#1.}%
  \whileboolexpr{test {\ifnumless{\theTABalignmentindex@}{#2}}}{%
    \stepcounter{TABalignmentindex@}%
    \edef\tabular@settings{\expandafter\@gobble\tabular@settings.}%
  }%
  \expandafter\@getnextTABchar\tabular@settings\\% GET NEXT TAB ALIGNMENT
  \if l\@nextTABchar\edef\stackalignment{l}\else%
    \if r\@nextTABchar\edef\stackalignment{r}\else%
      \if c\@nextTABchar\edef\stackalignment{c}\fi%
    \fi%                                     IGNORE IF NOT l, c, OR r
  \fi%
}

\def\@getnextTABchar#1#2\\{\gdef\@nextTABchar{#1}}
```

```
\newcommand\fixTABwidth[1]{%
  \if T#1\toggletrue{fixed@TABwidth}\else\togglefalse{fixed@TABwidth}\fi}

\newcommand\ensureTABstackMath[1]{%
  \let\sv@TABmode\TAB@delim\TABstackMath#1\let\TAB@delim\sv@TABmode}

\setstackEOL{\\}%              DEFAULT ROW SEP
\setstackTAB{&}%               DEFAULT COL SEP
\def\TAB@mathstyle{}%          NOTHING ADDED TO DEFAULT TAB MATH STYLE
\def\TAB@textstyle{}%          NOTHING ADDED TO DEFAULT TAB TEXT STYLE
\def\TAB@delim{}\TABstackText% INITIALIZE DEFAULT TO TEXT TABSTACKING
\TABunaryLeft%                 NO DEFAULT EMPTY {} GROUP AT LEFT END OF EACH CELL
\def\tabbed@gap{0pt}%          DEFAULT TABBED COL GAP
\def\align@gap{1em}%           DEFAULT ALIGN COL GAP
\def\tabular@gap{\tabcolsep}%  DEFAULT TABULAR COL GAP
\fixTABwidth{F}%               DEFAULT NON-FIXED WIDTH COLUMNS

\endinput
```