

# The **stackengine** Package

Highly customized stacking of objects, insets, baseline changes, *etc.*

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

July 11, 2013  
v2.0

Note: A syntax change in version 2.0 may require minor changes in your existing V1.0 stackengine code. See section 13 for details.

## 1 Definitions and Terms

Before I describe the various commands of this package, it is best for you to familiarize yourself with a few terms as used in this package.

### 1.1 Short Stacks and Long Stacks

In this package, “Short” refers to squeezing the extra space out of glyphs comprising the stack, so that the inter-item gap is solely what is set. Likewise, “Long” refers to a stack kept at constant inter-item baseline spacing. Let’s look at the following two examples to see this clearly.

t  
y  
k  
e

Notice the consistent inter-item gap on a short stack: e

t  
y  
k

Notice the consistent inter-baseline spacing on a long stack: e

In this package, the default spacing (*i.e.*, stacklength) for the long stack, saved

in the length `\Lstackgap` is the value `\baselineskip`. For short stacks, the default stacklength, stored in `\Sstackgap`, is 3pt, which corresponds to the same spacing found in T<sub>E</sub>X's `\shortstack` command. The default stack type constructed by this package is short, denoted by the definition of `\stacktype` as “S” (as opposed to “L” for long stacks). These defaults can be changed.

## 1.2 Baseline

The baseline is the horizontal line on which the current (unstacked) text is set. For non-descending letters, the bottom of the letters sit upon the baseline. In this document, we will graphically use the symbol  $\text{—}\mathbb{B}\text{—}$  to denote the current baseline of the text, as in

$\text{—}\mathbb{B}\text{—}$ A $\text{—}$ text continues here  
           B  
           C

## 1.3 (Over)Stacks and Understacks

In general terms, a stack is built up from the baseline, while an understack is built down from the baseline.

                  a  
                   b  
 So, for example, c is a stack, while a is an understack.  
                           b  
                           c

With this package, both stacks and understacks can be long or short, depending on how the invocation is made. Sometimes, in this documentation, a stack will be called an “overstack” in order to avoid confusion with “understack.”

## 1.4 Anchors

The term “anchor” is used to denote an item in the stack whose baseline does not change in the course of the stacking operation. In essence, stacking is done relative to the anchor.

                  a  
                   b  
 For example, in c, the letter “c” is the anchor.

In the understack a, “a” is the anchor.  
                   b  
                   c

## 1.5 Alignment

Anyone familiar with L<sup>A</sup>T<sub>E</sub>X knows something about the horizontal alignment settings “l”, “c” and “r”. As pertains to a stack, we refer to the alignment of the various rows of the stack, as shown in the following examples.

	abcd		abcd		abcd
	abc		abc		abc
	ab		ab		ab
Left:	a	Center:	a	Right:	a

The default alignment within a stack may be set with the `\stackalignment` definition, to possible values of `l`, `c`, or `r`, denoting left, center, and right alignments, respectively. In addition, a number of the stacking macros provide the alignment as an argument or optional argument that can be specified at time of invocation.

The `stackengine` package defaults to center alignment, unless reset by way of the `\stackalignment` definition.

## 2 Defaults and Macros Provided by `stackengine`

### 2.1 Defaults and How to Set Them

There are several macros that serve to store package default definitions. When a macro does NOT explicitly ask for one of the quantities defined by these defaults, it will use the values of these parameters to serve as the default value. However, some stacking macros may ask the user to specify an explicit value for any number of these parameters in the calling arguments. In that case, an explicit value may be provided or else the variable holding the default value may itself be provided as the argument.

Let us first consider the `stackgap` defaults, associated with short and long stacks. Those `stackgaps` are actually stored in the `\def`'s

```
\Sstackgap  
\Lstackgap
```

The parameter `\Sstackgap` is the default inter-item gap for any stacking command that builds a short stack. It defaults to 3pt. Correspondingly, `\Lstackgap` is the inter-item baselineskip for any stacking command that builds a long stack. It defaults to `\baselineskip`. To change these defaults, the following macro is provided:

```
\setstackgap{S}{inter-item stackgap}
\setstackgap{L}{inter-item baselineskip}
```

In addition, there is another macro, which may be invoked by the user:

```
\stackgap
```

The macro `\stackgap` will look at the current value of `\stacktype` and will output either `\Sstackgap` or `\Lstackgap`, depending on whether `\stacktype` is defined as “S” (short) or “L” (long).

In addition to these lengths, there are several parameter definitions that define default behaviors for those stacking macros when explicit values of these settings are not requested. The parameters, each set with a `\def` command, are

```
\def\stackalignment{l or c or r}
\def\quietstack{T or F}
\def\useanchorwidth{T or F}
\def\stacktype{S or L}
\def\usestackstrut{T or F}
```

The parameter `\stackalignment` defaults to “c” (center) and defines the default stacking alignment. Other options include “l” (left) and “r” (right). See §1.5 for more discussion of this parameter.

The parameter `\quietstack` defaults to “F” (false) and determines whether the result of a stacking operation is suppressed from the output. If the output is suppressed by setting this parameter to “T”, the most recent stacking operation may be recovered with a `\usebox{\stackedbox}`. See §3.1 for more discussion of this parameter.

The parameter `\useanchorwidth` defaults to “F” (false) and indicates that, upon creation, the stack width should be taken as the widest element of the stack. When set to “T”, the stack width, upon creation, is instead taken as the width of the anchor. Note that this parameter applies at the time of stack creation and cannot be used to retroactively change the characteristics of an existing stack. See §8 for more discussion of this parameter.

The parameter `\stacktype` defaults to “S” (short stack) and defines the default type of stack to build. When set to “L”, newly created stacks are, by default, long stacks. Refer to §1.1 for a definition of these terms.

The parameter `\usestackstrut` defaults to “T” (true) and has limited application. It only comes into play when using `\stackanchor` with `\stacktype` “S”. It is described in more detail in §6.1.

For macros which require explicit specification of some of these parameters,

the parameter token itself (*e.g.*, `\stackalignment`, `\quietstack`, *etc.*) may be provided as the argument, if the default behavior is desired.

## 2.2 Stacking Macros

### 2.2.1 `\stackengine`

The basic (and most general) stacking macro provided by this package is `\stackengine`. Nearly all other macros of this package merely provide an abbreviated invocation form of this macro, tailored for a particular application. The syntax is:

```
\stackengine{\Sstackgap or \Lstackgap or \stackgap or stacklength}
              {anchor}
              {item}
              {O or U}
              {\stackalignment or l or c or r}
              {\quietstack or T or F}
              {\useanchorwidth or T or F}
              {\stacktype or S or L}
```

There are no optional arguments with `\stackengine`. The first item is a stacking length, which can mean different things depending on whether a short- or long-stack is being created, as discussed in §1.1. The second argument is the stack anchor, whose baseline does not change. The third argument is the item stacked relative to the anchor. The fourth argument is either an “O” (for a normal or overstack) or a “U” (for an understack). The fifth argument denotes a left, center, or right alignment of the stack items with the use of “l”, “c”, or “r”. The sixth argument is either a “T” or “F” to denote whether the resulting stack is NOT printed (“T” denotes “do NOT print”). The seventh argument, also a “T” or an “F”, denotes when “T” that the width of the whole stack is to be taken as the width of the anchor. Otherwise, the width of the stack conforms to the width of the widest item in the stack. The final argument is defined as either an “S” or an “L” to denote whether a short stack or a long stack is being requested.

While none of the arguments to `\stackengine` have default values, there are various definitions in the package that store default values. When one wishes a stack to possess certain default properties, these default definitions may be passed as the values for the respective arguments to `\stackengine`. Arguments 1, 5, 6, 7, and 8 each possess a corresponding parameter containing the default variable. So, for example,

```
\stackengine{\stackgap}{A}{BC}{O}{\stackalignment}
              {\quietstack}{\useanchorwidth}{\stacktype}
```

will stack the letters “BC” over “A” using the default values of `\stackalignment`, `\quietstack`, `\useanchorwidth`, and `\stacktype`. In addition, the `stackgap` will be set to `\Sstackgap` if `\stacktype` is “S” or to `\Lstackgap` if `\stacktype` is “L”. Alternately, the same stack could be forced to be a long stack with right-alignment and be typeset using

```
\stackengine{\Lstackgap}{A}{BC}{0}{r}{F}{\useanchorwidth}{L}
```

### 2.2.2 `\stackon` and `\stackunder`

The `\stackon` and `\stackunder` macros are abbreviated forms of `\stackengine` in which the default values of parameters are taken. The only exception is the `stacklength`, which may be provided as an optional argument.

```
\stackon[stacklength]{anchor}{item }
\stackunder[stacklength]{anchor}{item }
```

In the case of `\stackon`, an (over)stack is created, whereas an understack is created with `\stackunder`. With both of these commands, the anchor is the first mandatory argument, and the item to be stacked above or below it is the second mandatory argument.

### 2.2.3 `\Shortstack` and `\Longstack`

The `\Shortstack` and `\Longstack` commands are stacking commands that allow more than two stacking rows to be specified at once. The items of a stack are provided in a space-separated list, as follows:

```
\Shortstack[alignment]{item {it em} {\item} $item$ ... anchor}
\Longstack[alignment]{item {it em} {\item} $item$ ... anchor}
```

The first item is at the top of the stack, farthest from the baseline, while the last item is the anchor of the stack. If an item (the contents of a single row) contains spaces, enclose the item in braces. If the item ends with a macro, enclose the item in braces. While not particularly intended for math mode (see §12 for more details), math items may be enclosed within dollar-sign delimiters. **Note that, because of the parsing algorithm, macros in the argument of `\Shortstack` and `\Longstack` may need to be `\protect`’ed.**

The alignment may be specified as an optional argument, with other parameters taken by their defaults. The `stacklength` is taken as `\stackgap`, which will depend on the value of `\stacktype`.

To give an example of the braced syntax described above, the stack

a  
b c  
¶  
 $\beta^2$

`\Shortstack[r]{a {b c} {\P} $\beta^2$ ANCHOR}` will produce ANCHOR.

#### 2.2.4 `\Shortunderstack` and `\Longunderstack`

The `\Shortunderstack` and `\Longunderstack` commands are the understacking equivalents of `\Shortstack` and `\Longstack`.

`\Shortunderstack[alignment]{anchor item {it em} {\item} $item$ ...}`  
`\Longunderstack[alignment]{anchor item {it em} {\item} $item$ ...}`

The caveats to the specification of the space-separated list of items is the same as that mentioned above for `\Shortstack` and `\Longstack`.

Note that the items are still specified from the top downward. In the case of understacks, however, that means that the anchor is the first item specified, while the last item in the list is farthest below the baseline.

#### 2.2.5 Top and bottom lapping macros

Top

and bottom lapping is achievable through the creation of various long stacks. The bottom syntax is the same for all six of the macros provided. The six names differentiate whether the lap is to the top (t) or the bottom (b) of the calling point, and whether it should appear to the left (l), centered (c), or to the right (r) of the calling location. The long-stacklength may be provided as an optional argument.

`\tllap[stacklength]{item}`  
`\tclap[stacklength]{item}`  
`\trlap[stacklength]{item}`  
`\bllap[stacklength]{item}`  
`\bclap[stacklength]{item}`  
`\brlap[stacklength]{item}`

See §5 for more information and examples of use for these lapping commands. If trying to remember the names of all these lapping commands is too tedious for you, two functionally identical macros have been introduced:

`\toplap[stacklength]{lap H-direction}{item}`  
`\bottomlap[stacklength]{lap H-direction}{item}`

The names are simpler to remember, but they require, as their first mandatory argument, the horizontal direction of lapping (l, c, or r), while the material to be lapped is now in the second mandatory argument.

### 2.2.6 `\stackanchor`

The `\stackanchor` macro is one way to create a stack where the baseline is split between two items in the stack. The syntax is

```
\stackanchor[stackgap]{top item}{bottom item}
```

The stacking gap may be provided as an optional argument. All other parameters taken are the defaults. There are subtle differences in the way a short stack anchor versus a long stack anchor is split. One should consult §6.1 for a description of how the stack is constructed.

If a short stack is being created with a split anchor, the `\usestackstrut` parameter will determine whether the split occurs at the mid-height of the strutbox (“T”) or whether it will occur about the baseline (“F”).

If a long stack is to be created with a split anchor, it is best to formulate and `\savestack` both the upper portion (with a stack) and the lower portion (with an understack), and then `\stackanchor` the two parts together.

### 2.2.7 `\abovebaseline` and `\belowbaseline`

The macros `\abovebaseline` and `\belowbaseline` also creates the means to shift an item’s baseline. The syntax is

```
\abovebaseline[stackgap]{item}  
\belowbaseline[stackgap]{item}
```

where the *stackgap* defines the stackgap distance. In a long stack, it will define the upward (for `\abovebaseline`) or downward shift (for `\belowbaseline`) of the baseline of the item. In a short stack, it will define the upward/downward vertical gap between the original baseline and the bottom/top of the shifted item.

For long stacks, a negative stackshift for `\belowbaseline` will produce the same result as the corresponding positive stackshift for `\abovebaseline`. But this will not be the case for short stacks. One should consult §6.2 for more details.

This command can also be employed to redefine the baseline of graphical objects, which can provide a useful technique to achieve horizontal alignment with other



objects. Negative stack gaps are valid (see §6.2 for examples).

### 2.2.8 `\topinset` and `\bottominset`

The macros `\topinset` and `\bottominset` are for insetting smaller items (*e.g.*, images) inside of a larger background items (*e.g.*, images). The syntax is

```
\topinset{inset item}{background item}{V-offset}{H-offset}  
\bottominset{inset item}{background item}{V-offset}{H-offset}
```

In addition to providing the inset and background items as the first two arguments, the inset may be vertically and horizontally offset through the use of the last two arguments. The vertical offset is relative to the top of the `\topinset` background and relative to the bottom of the `\bottominset` background. The horizontal offset is relative to the left edge of the background for left stackalignment and relative to the right edge of the background for right stackalignment. Horizontal offset is meaningless for center stackalignment.

While these commands were designed with inlaid images in mind (see §9), there is nothing that prevents their use to inlay character glyphs upon each other, as in  $\Theta$ , accomplished with `\topinset{*}{0}{0.2ex}{0.0ex}`.

### 2.2.9 `\savestack`

The macro `\savestack` provides a means to save an intermediate result (in a box that can be recalled with the specified macro) without printing it. Its syntax is

```
\savestack{macro token}{stacking operation}
```

The macro token is a backslashed name that you can later use to recall the boxed intermediate result. The stacking operation is the intermediate result you wish to save in a box without printing. Reasons why this might be a useful macro are described in §3.2.

Note that while `\savestack` is designed to save an intermediate stacking operation in a box, the reality is that you can use this macro to save any sequence of boxable output, even if it is not a stacking operation. So, for example, `\savestack{\mymacro}{\fbox{A}}` can later be recalled with an invocation of `\mymacro`, to yield  $\boxed{A}$ .

## 3 Making a Stack in Stages

There are times where one needs to make a stack in several consecutive steps. In order to do so, there are several ways, detailed below.

### 3.1 Suppressing Output

One can define `\quietstack` as `T` to suppress output. Then, the intermediate stack can be passed as an argument to the next stacking operation with a `\usebox{\stackedbox}`. That same `\usebox` may be used to output the final stack, as long as `\quietstack` remains defined as “`T`”. Otherwise, `\quietstack` must again be defined “`F`” (false) to re-enable automatic output.

### 3.2 Saving a Stack without Printing It

Perhaps a quicker and better way is to use the package’s `\savestack` macro, of the form `\savestack{macro token}{stacking operation}`. This will perform the stacking operation and place it into the supplied token (without printing it), so that this intermediate stack can be passed to the subsequent stacking operation by executing the token macro.

For example,

```
\savestack{\pdq}{\Longstack[r]{6 5 4 {3 3}}}%
\Longstack[l]{\pdq} 2 1 0}%
```

6  
5  
4

would first place “`3 3`” into `\pdq`, without printing it, in a right-aligned fashion, with the “`3 3`” on the baseline. Then, the second `\Longstack` would place the first stack atop of the “`2`” in a left-aligned fashion. The resulting stack would look as follows, with the “`0`” on the final baseline.

6  
5  
4  
3 3  
2  
1  
0 ——— B—

## 4 Choosing the Anchor Element

All stacking done with the stackengine has a designated anchor element. The baseline of the anchor does not change. Instead, the other items are placed relative to the anchor. By adding items at first above and then below the anchor, any arbitrary stack can be built around the anchor.

3  
2  
1

Here is a Longstack/understack example: 0 that was achieved as follows:

-1  
-2  
-3  
-4  
-5

```
\savestack{\pdq}{\Longstack{3 2 1 0}}%
\Longunderstack[r]{\{\pdq} -1 -2 -3 -4 -5}
```

In the first stack, “0” is the anchor of `\pdq`. In the second stack, `\pdq` is the anchor (which means that “0” remains the anchor of the composite stack).

## 5 Top and Bottom Lapping

Top and bottom lapping can be accomplished with a long stack, using a null item as the anchor, and `\useanchorwidth` set to `T`. This technique is encoded in six `\xylap` commands, where  $x$  can be `t` or `b` (for “top” and “bottom,” respectively) and  $y$  can be `l`, `c`, or `r` (for “left,” “center,” and “right,” respectively). Note that this  $y$  is not the same as the `\stackalignment`, but rather refers to the direction where the lapped item is placed relative to the lap invocation.

For illustration only, laps will be shown at a *visible* lapmark (given by `|`), defining the point of invocation. In these examples, I lap a `\parbox` above or below the selected location.

<p>this is a top-left lap <code>\tllap</code></p>	<p>this is a top-right lap <code>\trlap</code></p>
<p>For example here  or else here, or else I can put it here  or here .</p>	
<p>this is a bottom-left lap <code>\bllap</code></p>	<p>this is a bottom-right lap <code>\brlap</code></p>

this is a top-center lap <code>\tclap</code>
---

Of course, I can perform center lapping, as well. To see, for example, how the

this is a bottom- center lap <code>\bcclap</code>
--

above was achieved, the last lap in the prior sentence was achieved by invoking

```
\tclap{\fbox{\parbox[b]{1.1in}{this is a top-center lap \tcl}}}
```

in the middle of the word “well.” (Note that `\tcl` is a box containing the verbatim text “`\tclap`,” created with the `verbatimbox` package.)

Note that `\toplap` and `\bottomlap` have also been introduced as macros that are functionally identical to these six lapping commands. While requiring an extra argument, their names may be easier for you to remember. See section 2.2.5.

## 6 Shifting the Anchor’s Baseline

By default, the baseline of the anchor becomes the baseline of the stack. If one wanted the stack baseline to **not** conform to the baseline of any of the individual stacked items, something needs to change. If one quantitatively knows the shift desired, a simple `\raisebox` could be used to vertically shift the anchor, after (or possibly prior to) stacking. There are also some other options provided by the package.

### 6.1 The `\stackanchor` Macro for “L” and “S” Stack Types

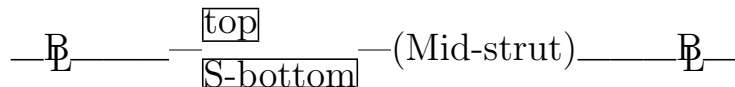
If one would like the anchor’s baseline to be “midway” between two vertically arrayed items, the `\stackanchor` macro attempts to provide that capability for both “L” and “S” stack types, though the definitions of “midway” are different for each. The macro `\stackanchor` works for two-item stacks in both “L” and “S” stack types, as shown below.

In the case on a long stack, what is satisfied is that the baseline of the anchor is made equidistant from the baselines of the two respective items comprising the stacked anchor.

— $\mathbb{P}$ —	— $\overline{\hspace{1cm}}$ —	— $\mathbb{P}$ —
	top	
	— $\overline{\hspace{1cm}}$ —	
	L-bottom	

In the case of a short stack, the middle of the inter-item gap is vertically located at the center of the font’s `\strutbox`, so that the middle of the stackgap would be at the same vertical level as the middle of parentheses, were they located on

the original baseline.

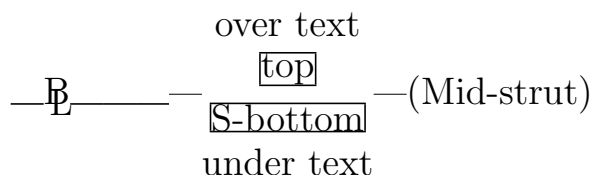


If one does not want to have the short-stack gap split at the mid-strut height, but rather at the baseline, the parameter `\def\usestackstrut{F}` may be set prior to the `\stackanchor` invocation.



The `\usestackstrut` parameter will only affect short stacks (`\stacktype "S"`), since long stacks do not use a strut as part of the `\stackanchor` definition.

For stackanchors with more than two rows, one should compose the top portion with a stack and the bottom portion with an understack, and then, finally, shift the anchor by applying `\stackanchor`.



## 6.2 `\belowbaseline` for Moving the Anchor's Baseline

The command `\belowbaseline` is intended to be used to set an item, such as a stack anchor, below the baseline (this discussion is equally valid for the macro `\abovebaseline`, but in the opposite sense). The macro accomplishes this task by stacking the item below a null object. However, `\useanchorwidth` is temporarily set to F so that the stacked object takes on the width of the under-placed object.

The command can be used with both stacktypes. With `\stacktype "S"`, the optional argument will denote the stackgap below the baseline (default `\Sstackgap`),



When used in with a `\stacktpe "L"`, the optional length denotes the distance to the baseline of the understacked object (default `\Lstackgap`).

Long stacktype  $\xrightarrow{\boxed{0\text{pt below baseline}}}$   $\xrightarrow{\boxed{-18\text{pt}}}$   $\xrightarrow{\boxed{18\text{pt}}}$

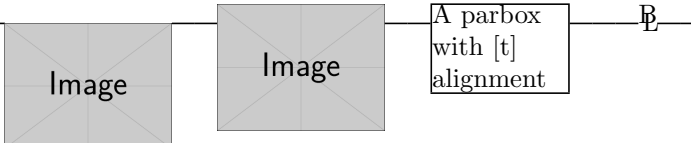
As the above images show, there is nothing to prevent the use of negative numbers being used to raise the the object above the baseline. But care must be taken, since a `\belowbaseline` with a negative shift will not necessarily equal the `\abovebaseline` with the corresponding positive shift (that equality only holds for long stacks). To better appreciate the meaning of negative baseline shifts, we compare, below, negative shifts for `\belowbaseline` to `\abovebaseline` with the corresponding positive argument.

Short stacktype  $\xrightarrow{\boxed{-6\text{pt below}}}$   $\xrightarrow{\boxed{6\text{pt above}}}$   $\mathbb{P}$

Long stacktype  $\xrightarrow{\boxed{-6\text{pt below}}}$   $\xrightarrow{\boxed{6\text{pt above}}}$   $\mathbb{P}$

The `\belowbaseline` command is also useful for forcing objects, such as images, to be top-aligned. This can be a useful feature when trying to achieve a desired horizontal alignment of disparate elements in tabular columns.

```
\def\stacktype{S}\belowbaseline[0pt]{\imgi}
\belowbaseline[0pt -\heightof{\strutbox}]{\imgi}:
```

$\mathbb{P}$    $\mathbb{P}$

The diagram shows a horizontal baseline with a  $\mathbb{P}$  symbol at each end. Between the symbols are three elements: a gray square labeled 'Image', another gray square labeled 'Image', and a white rectangle with a black border labeled 'A parbox with [t] alignment'. All three elements are aligned to the top of the baseline.


## 7 Negative Stack Gap

A negative gap can be used to overlap things. For example,  $\bullet$  plus “i” can make, when the stack gap is suitably negative, the following:

$\bullet$   
i

Note that the use of a negative stack gap for a two-item stack is actually very similar to the inset commands described in §2.2.8 and §9. The issue of negative stackgaps is also discussed in section 6.2.

## 8 The `\useanchorwidth` Option

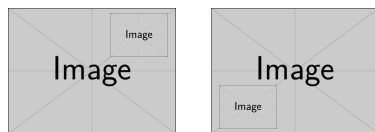
Because `•` has width larger than the dot glyph itself (we box its extent here: ) , stacking it could otherwise be a problem. For example, `th i s` uses the whole stackwidth to set the dotted-i in its place. On the other hand, when `\useanchorwidth` is defined as `T`, “`this`” uses only the anchor width to set the stack (where the “i” is the anchor).

Below, we show an `\fbox` around the dotted i, clearly demonstrating that the dot does not influence the width of the stack when `\useanchorwidth` is defined as “`T`”.



## 9 Image Insets

The commands `\topinset` and `\bottominset` allow images to be stacked atop one another, to produce the effect of an inset image. In addition to specifying the two images, the effect can be tailored by specifying the vertical and horizontal offset of the inset image.



Note that the horizontal offset only works when `\stackalignment` is set to `l` or `r`.

The last figure was obtained with `\bottominset{\imgii}{\imgi}{2pt}{3pt}`, showing that `\imgii` was offset 2pt from the bottom and 3pt from the left edge of `\imgi`.

## 10 Stacking for Subfiguring and Baseline Manipulation

The stacking commands provide a convenient way to underset a subfigure notation under the corresponding image. Furthermore, because of the flexibility of the stacking commands, the baselines of the subfigures can be manipulated

in many ways, which may come in handy depending on how you are choosing to lay out your subfigures.

Examples of this are shown in figure 1. Note from the definition used for fig-

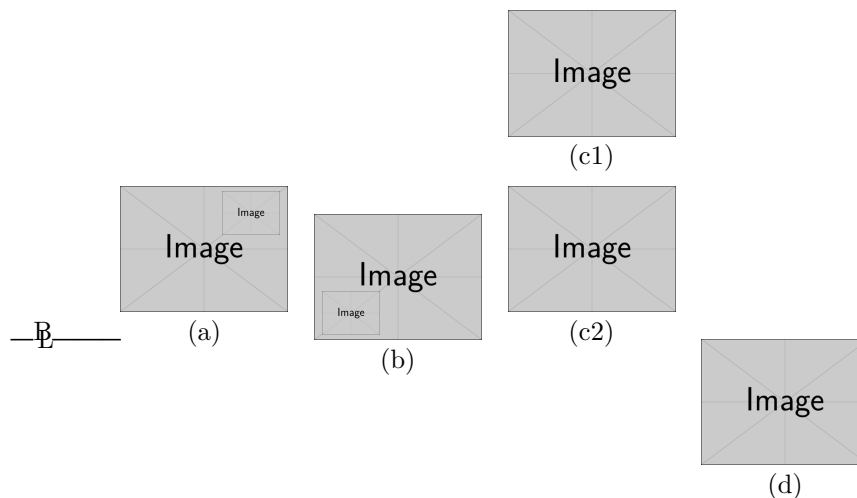


Figure 1. Image stacking with stacktype “S”, stack gap set to 3pt and using:  
 (a) `\stackon{(a)}{\imga}`, (b) `\stackunder{\imgb}{(b)}`,  
 (c) `\Shortstack{{\protect\imgi} (c1) {} {\protect\imgi} (c2)}`,  
 and (d) `\belowbaseline[0pt]{\stackunder{\imgi}{(d)}}`.

ure 1(c) that when a `\savestack` has been used to place a graphic into a token (e.g., `\imgi`) that it must be protected when used inside a space-separated argument, such as `\Shortstack`, `\Longstack`, and the corresponding understacks.

## 11 Nested Stacking Commands

The macros `\Shortstack`, `\Longstack`, and the corresponding understack macros are set up to conveniently stack multiple objects using a space separated argument list. Because of parsing constraints on the space-separated list, however (see §12), there may be times that you would prefer building stacks by way of nesting the more primitive stacking macros.

Nesting can be accomplished, but some care must be taken, in the case of long stacks. But first, let us see the ways in which nesting may be applied for short stacks:



```

\stackunder{1}{\stackunder{2}{\stackunder{3}{\stackunder{4}{5}}}}
\stackunder{\stackunder{\stackunder{\stackunder{1}{2}}{3}}{4}}{5}
\stackon{1}{\stackon{2}{\stackon{3}{\stackon{4}{5}}}}
\stackon{\stackon{\stackon{\stackon{1}{2}}{3}}{4}}{5}

```

will produce the following:

$$\begin{array}{c}
 5\ 5 \\
 4\ 4 \\
 3\ 3 \\
 2\ 2 \\
 1\ 1\ 1\ 1\ \underline{\text{P}}\text{---} \\
 2\ 2 \\
 3\ 3 \\
 4\ 4 \\
 5\ 5
 \end{array}$$

In these cases, the nesting could be applied on either the first or the second argument of the `\stackunder` or `\stackon` commands, with the same end result.

In the case of long stacks however, employing the identical methodology gives

$$\begin{array}{c}
 5 \\
 4 \\
 3 \\
 2\ \mathbf{3} \\
 1\ 1\ 1\ 1\ \underline{\text{P}}\text{---} \\
 2\ \mathbf{3} \\
 3 \\
 4 \\
 5
 \end{array}$$

a different, at first unexpected, result:

In this case, nesting the second argument works as hoped, but nesting the first argument does not. The reason for this “failure” stems not from a failure of logic, but an enforcement of it. It stems from the very definition of a long stack, in which the baseline of the second argument is placed `\LStackgap` below or above the baseline of the first argument. By nesting the first argument, all the second arguments are placed `\Lstackgap` from the unchanging stack baseline (thus overlapping), and not relative to the current top or bottom of the nested stack.

Therefore, to nest long stacks, only the second argument of the `stackengine` may be nested.

## 12 Math Mode Usage

The `stackengine` package is not really intended for math mode, since there are many packages that already cater directly to the need to stack and align mathematical objects. However, if there is a need to use `stackengine` there, because of its particular facility with stacking gaps, there are several hints to remember.

1. Stacking commands from this package can be used in math mode, but will set their arguments as text, not math.

$$\backslash ( y = \backslash \text{stackunder}\{a\}{+} x \backslash ) : \quad y = \frac{ax}{+}$$

2. To set an argument in math style, it must be set between math delimiters.

$$\backslash ( y = \backslash \text{stackunder}\{\backslash (a\backslash )\}{+} x \backslash ) : \quad y = \frac{qx}{+}$$

3. Each space-separated item of the argument to `\Shortstack`, `\Longstack`, and the corresponding understacks must each be enclosed in their own set of dollar-sign math delimiters (Because of the way the argument is parsed, the `\(` and `\)` delimiters can be used, but only if `\protect`'ed, in these space-separated argument lists).

$$\backslash ( y = \backslash \text{Shortstack}\{\$a\$ . \$b\$ \} x \backslash ) : \quad y = \frac{a}{b}x$$

4. As described in §2.2.3, if the math argument has spaces in a command which uses space-separated lists, the argument must be enclosed in braces.

$$\backslash ( y = \backslash \text{Shortstack}\{\{$a + b\$ \} . \$b\$ \} x \backslash ) : \quad y = \frac{a+b}{b}x$$

5. For more complex items, the `\savestack` command can be used to save an intermediate math expression (even if it isn't actually a stack).

$$\begin{array}{l} \backslash \text{savestack}\{\backslash a\}\{\backslash (\backslash \text{sqrt}\{a + b\}\backslash )\} \\ \backslash \text{savestack}\{\backslash \text{dash}\}\{- - - - \} \\ \backslash ( y = \backslash \text{stackunder}\{\backslash \text{stackon}\{\backslash \text{dash}\}\{\backslash a\}\}\{\$b\$ \} x \backslash ) : \end{array} \quad y = \frac{\sqrt{a+b}}{b}x$$

6. In math mode, the `\stackanchor` command seems to preserve the math axis better with an “S” `\stacktype`, rather than type “L”.

## 13 Backward Compatibility

The author of this package profoundly apologizes for the fact that the syntax of this package has changed between versions 1.0 and 2.0 of this package.

In version 1.0 of this package, stackgaps could be directly defined by setting the L<sup>A</sup>T<sub>E</sub>X lengths `\Sstackgap` and `\Lstackgap`. The problem with this approach is that, under fontsize changes, these stackgaps were not automatically updated to reflect the new fontsize. As of V2.0, that deficiency was remedied. The syntax that replaces these length specifications is

```
\setstackgap{S}{inter-item stackgap}
\setstackgap{L}{inter-item baselineskip} .
```

Thus, `\Sstackgap` and `\Lstackgap` are no longer lengths (but instead `\defs`), and cannot be set via the `\Sstackgap=...` or `\setlength{\Sstackgap}{...}` syntax. It is these settings which will need to be revised to modernize older code.

While not encouraged, the V1.0 syntax may be recovered via

```
\usepackage[oldsyntax]{stackengine}
```

If you retain the “oldsyntax” setting, you must remember that `stackengine` will honor your specified gap sizes (`\Sstackgap` and `\Lstackgap`) at the time of their **definition**, rather than at the time of their **invocation**. Thus, if `\Sstackgap` is specified as `0.7ex` when the text is `\normalsize`, the gap length is set in a `\normalsize` font, even if the fontsize is later changed. To overcome this behavior, the stackgap must be redefined at the new fontsize, or better still, `[oldsyntax]` usage needs to be relegated.

## 14 Acknowledgements

I would like to thank Prof. Enrico Gregorio at [tex.stackexchange](http://tex.stackexchange.com) for providing three lines of code to strip a leading backslash from an argument:

```
http://tex.stackexchange.com/questions/42318/
removing-a-backslash-from-a-character-sequence
```

Furthermore, Prof. Gregorio was of great assistance in making the transition from the deficient V1.0 syntax as painless as possible:

```
http://tex.stackexchange.com/questions/123443/
defining-a-length-that-scales-with-fontsize-changes/123470#123470
```

## 15 Code Listing

```
\def\stackengineversionnumber{v2.0}
\ProvidesPackage{stackengine}
[2013/07/11 \stackengineversionnumber
  Stacking text and objects in convenient ways]
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status ‘maintained’.
%
```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\fi
\def\useanchorwidth{F}% T= FORCE BOX WIDTH TO ANCHOR-WIDTH
\def\quietstack{F}% T = SUPPRESS OUTPUT OF ALL COMMANDS
% (USE \usebox{\stackedbox} TO RETRIEVE)
\def\stackalignment{c}% l, c, or r for left, center or right alignment
\def\stacktype{S}% S for constant gap shortstack;
% L for constant baselineskip longstack
\def\usestackstrut{T}%

%\stackengine{S\stackgap or \Lstackgap or \stackgap or stacklength}
% {anchor}
% {item}
% {O or U}
% {\stackalignment or l or c or r}
% {\quietstack or T or F}
% {\useanchorwidth or T or F}
% {\stacktype or S or L}
\newcommand*\stackengine[8]{%
  \sbox{\stackedbox}{%
    \sbox{\anchorbox}{#2}%
    \sbox{\addedbox}{#3}%
    \settowidth{\stackedboxwidth}{#2}%
    \if F#7%
      \ifdim\wd\addedbox>\stackedboxwidth%
        \settowidth{\stackedboxwidth}{#3}%
      \fi%
    \fi%
    \setlength{\boxshift}{#1}%
    \if L#8%
      \if U#4%
        \setlength{\boxshift}{-\boxshift}%
      \fi%
    \else%
      \if U#4%
        \setlength{\boxshift}{-\dp\anchorbox -\ht\addedbox -\boxshift}%
      \else%
        \setlength{\boxshift}{\ht\anchorbox +\dp\addedbox +\boxshift}%
      \fi%
    \fi%
    \if c#5%
      \hspace{.5\stackedboxwidth}%
      \hspace{-.5\wd\anchorbox}%
      \usebox{\anchorbox}%
      \hspace{-.5\wd\anchorbox}%
      \hspace{-.5\wd\addedbox}%
      \raisebox{\boxshift}{#3}%
      \hspace{-.5\wd\addedbox}%
      \hspace{.5\stackedboxwidth}%
    \else%
      \if l#5%
        \usebox{\anchorbox}%
        \hspace{-.5\wd\anchorbox}%
        \raisebox{\boxshift}{#3}%
        \hspace{-.5\wd\addedbox}%
        \hspace{.5\stackedboxwidth}%
      \else%

```

```

\if r#5%
\hspace{\stackedboxwidth}%
\hspace{-\wd\anchorbox}%
\usebox{\anchorbox}%
\hspace{-\wd\addedbox}%
\raisebox{\boxshift}{#3}%
\fi%
\fi%
\fi%
}%
\if F#6\usebox{\stackedbox}\fi%
}

\newcommand*\stackunder[3][\stackgap]{\stackengine%
{#1}{#2}{#3}{U}{\stackalignment}{\quietstack}{\useanchorwidth}{\stacktype}}

\newcommand*\stackon[3][\stackgap]{\stackengine%
{#1}{#2}{#3}{0}{\stackalignment}{\quietstack}{\useanchorwidth}{\stacktype}}

\newcommand*\Shortstack[2][\stackalignment]{\@stack{#1}{#2}{0}{S}}

\newcommand*\Longstack[2][\stackalignment]{\@stack{#1}{#2}{0}{L}}

\newcommand*\Shortunderstack[2][\stackalignment]{\@stack{#1}{#2}{U}{S}}

\newcommand*\Longunderstack[2][\stackalignment]{\@stack{#1}{#2}{U}{L}}

\newcommand*\@stack[4]{%
\getargsC{#2}%
\if U#3%
\setcounter{stackindex}{\narg}%
\def\@stacksign{-}%
\def\@stackcondition{\thestackindex > 1}%
\else%
\setcounter{stackindex}{1}%
\def\@stacksign{%
\def\@stackcondition{\thestackindex < \narg}%
\fi%
\sbox{\stackedbox}{\csname arg\roman{stackindex}\endcsname}%
\whiledo{\@stackcondition}{%
\addtocounter{stackindex}{\@stacksign 1}%
\stackengine{\csname#4stackgap\endcsname}%
{\csname arg\roman{stackindex}\endcsname}%
{\usebox{\stackedbox}}{#3}{#1}{T}{\useanchorwidth}{#4}%
}%
\if F\quietstack\usebox{\stackedbox}\fi%
}

% PERHAPS THESE SIX MACROS CAN BE REPLACED BY \toplap AND \bottomlap
\newcommand*\tllap[2][\Lstackgap]{\@stacklap{#1}{l}{#2}{0}}
\newcommand*\tlclap[2][\Lstackgap]{\@stacklap{#1}{c}{#2}{0}}
\newcommand*\trlap[2][\Lstackgap]{\@stacklap{#1}{r}{#2}{0}}
\newcommand*\bllap[2][\Lstackgap]{\@stacklap{#1}{l}{#2}{U}}
\newcommand*\bclap[2][\Lstackgap]{\@stacklap{#1}{c}{#2}{U}}
\newcommand*\brlap[2][\Lstackgap]{\@stacklap{#1}{r}{#2}{U}}

\newcommand*\toplap[3][\Lstackgap]{\@stacklap{#1}{#2}{#3}{0}}

```

```

\newcommand*\bottomlap[3][\Lstackgap]{\@stacklap{#1}{#2}{#3}{U}}

\newcommand*\@stacklap[4]{%
  \if l#2\stackengine{#1}{#3}{#4}{r}{F}{T}{L}\else%
    \if r#2\stackengine{#1}{#3}{#4}{l}{F}{T}{L}\else%
      \stackengine{#1}{#3}{#4}{c}{F}{T}{L}%
    \fi%
  \fi
}

\newcommand*\stackanchor[3][\stackgap]{%
  \setlength{\stack@tmplength}{#1}%
  \setlength{\stack@tmplength}{.5\stack@tmplength}%
  \if T\usestackstrut%
    \if S\stacktype\addtolength{\stack@tmplength}%
      {.5\ht\strutbox-.5\dp\strutbox}\fi%
    \fi%
  \stackengine{\stack@tmplength}{#2}{0}{\stackalignment}{T}{F}{\stacktype}%
  \if T\usestackstrut%
    \if S\stacktype\addtolength{\stack@tmplength}%
      {-\ht\strutbox+\dp\strutbox}\fi%
    \fi%
  \stackengine{\stack@tmplength}{\usebox{\stackedbox}}{#3}{U}%
    {\stackalignment}{\quietstack}{F}{\stacktype}%
}

\newcommand*\abovebaseline[2][\stackgap]{%
  \stackengine{#1}{#2}{0}{\stackalignment}{\quietstack}{F}%
    {\stacktype}}

\newcommand*\belowbaseline[2][\stackgap]{%
  \stackengine{#1}{#2}{U}{\stackalignment}{\quietstack}{F}%
    {\stacktype}}

\newcommand*\topinset[4]{\stack@inset{#1}{#2}{#3}{#4}{0}}

\newcommand*\bottominset[4]{\stack@inset{#1}{#2}{#3}{#4}{U}}

\newcommand*\stack@inset[5]{%
  \ifthenelse{\equal{#3}{}}%
    {\setlength{\stack@tmplength}{0pt}}%
    {\setlength{\stack@tmplength}{#3}}%
  \ifthenelse{\equal{#4}{}}{\def\stack@tmp{0pt}}{\def\stack@tmp{#4}}%
  \def\stack@lroffset{\rule{\stack@tmp}{0pt}}%
  \addtolength{\stack@tmplength}{\heightof{#1}}%
  \stackengine{-\stack@tmplength}{#2}{\stack@lroffset#1\stack@lroffset}%
    {#5}{\stackalignment}{\quietstack}{\useanchorwidth}{S}%
}

\newcommand*\savestack[2]{%
  \global\edef\sv@name{\stack@macro@name{#1}}%
  \@ifundefined{\sv@name content}{%
    \expandafter\newsavebox\expandafter{\csname\sv@name content\endcsname}%
  }{%
    \expandafter\sbox\csname\sv@name content\endcsname{#2}%
    \protected@edef#1{\usebox{\csname\sv@name content\endcsname}}%
  }
}

```

```
}  
\endinput
```