The delimset Package

Niklas Beisert

Institut für Theoretische Physik
Eidgenössische Technische Hochschule Zürich
Wolfgang-Pauli-Strasse 27, 8093 Zürich, Switzerland

nbeisert@itp.phys.ethz.ch

2025/03/25, v2.2.1

https://ctan.org/pkg/delimset

Abstract

delimset is a LaTeX 2ε package to typeset and declare sets of delimiters in math mode whose size can be adjusted conveniently.

Contents

1 Introduction

In ordinary TEX and LATEX, delimiters in math and physics expressions (brackets such as parentheses, braces, but also absolute values, sets, pairings, bra-kets and quantum expectation values, commutators, ...) are typically coded by their respective symbols. For example:

```
[(ax+b)x+c]                  → [(ax + b)x + c]
\{1,2,3,\ldots\}             → {1, 2, 3, ...}
(v,w)                        → (v, w)
\langle\psi|M|\psi\rangle    → ⟨ψ|M|ψ⟩
(x^2+px+q)|_{-p/2}           → (x^2 + px + q)|_{−p/2}
```

...

In order to adjust the size of delimiters, the modifiers `\big`, `\Big`, `\bigg`, `\Bigg` are used. More importantly the construct `\left*` ... [`\middle*`] ... `\right*` adjusts the size of delimiters to the contents:

$$\left[\left[\left[\left[[*]\right]\right]\right]\right], \qquad \left\{\frac{p}{q}\middle| p,q\in\mathbb{Z}, q\neq 0\right\}$$

These modifiers allow to construct expressions involving delimiters conveniently and with a large amount of flexibility.

However, once an expression has been typeset in this way, it takes some efforts to modify, if needed. At least, one has to take care of both delimiters at the same time and change their size or type accordingly. For example, $\hat{X}$ in the expression `(A\hat{X}+B)` has an extended height that calls for bigger delimiters. Ideally, one would use `\left(...\right)` to find the correct size. Unfortunately, in this case, the smallest size of delimiters that covers the height of $\hat{X}$ is equivalent to `\Big`, which feels somewhat too big. For aesthetic reasons one might thus prefer the size `\big(...\big)`:

```
(A\hat{X}+B)                 → (AX̂ + B)
\left(A\hat{X}+B\right)      → (AX̂ + B)
\Big(A\hat{X}+B\Big)         → (AX̂ + B)
\big(A\hat{X}+B\big)         → (AX̂ + B)
```

The package delimset provides mechanisms to declare sets of delimiters whose size can be adjusted conveniently by adding simple flag parameters. For example, it provides a general purpose bracket `\brk` which can be used as follows:

```
\brk{A\hat{X}+B}             → (AX̂ + B)
\brk*{A\hat{X}+B}            → (AX̂ + B)
\brk^2{A\hat{X}+B}           → (AX̂ + B)
\brk!{A\hat{X}+B}            → (AX̂ + B)
```

It also allows to change the type of bracket conveniently:

```
\brk{ax+b}                   → (ax + b)
\brk[s]{ax+b}                → [ax + b]
\brk[c]{ax+b}                → {ax + b}
\brk[a]{ax+b}                → ⟨ax + b⟩
```

These features can be combined and used, e.g., in nested brackets in order to distinguish the levels by shape and size:

$$\texttt{\textbackslash brk[s]!\{\textbackslash brk\{ax+b\}x+c\}} \to \big[(ax+b)x+c\big]$$

All of this can of course be achieved with the conventional tools of TEX with comparable effort, but the more complicated and nested the expressions get, the more difficult it will be to adjust them to obtain a visually acceptable result.

The main functionality of the package is based on a versatile mechanism to specify sets of delimiters, e.g.:

$$\texttt{\textbackslash delim<>\{ax+b\}} \to \langle ax+b\rangle$$
$$\texttt{\textbackslash delimpair[\{*,\}[!\{a\}\{b\}} \to \big[a,b\big[$$
$$\texttt{\textbackslash delimtriple<||>*\{\textbackslash psi\}\{A\^{}\textbackslash dagger\}\{\textbackslash psi\}} \to \big\langle\psi\big|A^\dagger\big|\psi\big\rangle$$

The command `\delim` can be used on the fly, but also in definitions of custom delimiter sets such as:

```
\newcommand{\comm}{\delimpair{[}{*,}{]}}
```

Note that the definition of `\comm` does not specify any arguments. They are nevertheless read by the incomplete definition of `\delimpair` from what follows `\comm`. In particular, this incomplete definition enables the correct parsing of the optional size specifier flag, e.g.:

$$\texttt{\textbackslash comm!\{P\}\{\textbackslash psi(x)\}} \to \big[P,\psi(x)\big]$$

The package also provides a mechanism to declare delimited sets more flexibly. The above definition could be written as follows:

```
\DeclareMathDelimiterSet{\comm}[2]
    {\selectdeliml{[}{#1},{#2}\selectdelimr{]}}
```

1.1 Delimiter Sizes for Math Styles

In plain LaTeX 2_ε the size modifiers `\big`, `\Big`, `\bigg`, `\Bigg` have the shortcoming that they are based on a fixed font size of 10pt. More precisely, they use a vertical phantom of height 8.5pt, 11.5pt, 14.5pt or 17.5pt, respectively to set the height of the delimiter. The package amsmath corrects for font size by instead placing a (centred) vertical phantom of height 1.2, 1.8, 2.4 or 3 times the size of the currently selected math font (height plus depth of `\Mathstrutbox@`).

Unfortunately, it does not account for the currently selected math style. Therefore, the size of delimiters in sub/superscripts cannot be adjusted appropriately (an arguably better typesetting practice is to avoid complicated expressions in sub/superscripts in the first place), they come out way too big:

$$\texttt{e\^{}\{\textbackslash big(ax+b\textbackslash big)\}} \to e^{\big(ax+b\big)}$$

This package modifies the definitions of the size modifiers (of amsmath) to automatically adjust to sub/superscripts (subject to availability in the font):

$$\texttt{e\^{}\{\textbackslash big(ax+b\textbackslash big)\}} \to e^{\big(ax+b\big)}$$

1.2 Spacing and Math Classes

Another shortcoming of the variable-size delimiters is that the spacing is noticeably different from their fixed-size counterparts:

$$\texttt{\square(ax+b)\square} \rightarrow \square(ax+b)\square$$
$$\texttt{\square\bigl(ax+b\bigr)\square} \rightarrow \square\bigl(ax+b\bigr)\square$$
$$\texttt{\square\left(ax+b\right)\square} \rightarrow \square\left(ax+b\right)\square$$
$$\texttt{\square\left(ax+\big.b\right)\square} \rightarrow \square\left(ax+b\right)\square$$

Often the construct `\left*` ... `\right*` leaves a large amount of space around it. A suitable way to fix this problem is to adjust the math class as follows:

$$\texttt{\square(ax+b)\square} \rightarrow \square(ax+b)\square$$
$$\texttt{\square\mathopen{}\mathclose{\left(ax+b\right)}\square} \rightarrow \square(ax+b)\square$$

This makes the expression look like `\mathopen` from the left and like `\mathclose` from the right. Importantly, the delimited expression should be contained in the `\mathclose` block so as to place any following sub/superscripts at the appropriate height. Unfortunately, the fix is too elaborate in comparison to the benefits of appropriate spacing. For practical purposes, consistent spacing can only be achieved by a more convenient mechanism.

On a related note, it is important to correctly specify the math class (such as `\mathopen` or `\mathclose`) for the delimiters, for example:

$$\texttt{\big(-1\big)} \rightarrow \big(-1\big)$$
$$\texttt{\bigl(-1\bigr)} \rightarrow \bigl(-1\bigr)$$

The math class can also make a major difference for intermediate delimiters, e.g.:

$$\texttt{\bigl<\psi\big|\psi\bigr>} \rightarrow \bigl\langle\psi\big|\psi\bigr\rangle$$
$$\texttt{\bigl<\psi\mathpunct\big|\psi\bigr>} \rightarrow \bigl\langle\psi\big|\,\psi\bigr\rangle$$
$$\texttt{\bigl<\psi\mathinner\big|\psi\bigr>} \rightarrow \bigl\langle\psi\,\big|\,\psi\bigr\rangle$$
$$\texttt{\bigl<\psi\mathbin\big|\psi\bigr>} \rightarrow \bigl\langle\psi\,\big|\,\psi\bigr\rangle$$
$$\texttt{\bigl<\psi\bigm|\psi\bigr>} \rightarrow \bigl\langle\psi\,\bigm|\,\psi\bigr\rangle$$

Depending on the particular situation, any of these expressions may be the most appropriate representation.

The package delimset automatically takes care of the math classes of the left and right delimiters. It also offers several choices for intermediate delimiters to take the context into account.

1.3 Compounds and Broken Lines

The mechanism provided by the package requires the delimited expression to appear as an uninterrupted compound. In particular, the expression cannot span several lines in a multi-line equation or several columns in a matrix. The same restriction applies to the elementary `\left` and `\right` construct. In cases where delimited expressions are torn apart across several blocks, the delimiters have to be typeset individually. The package offers a mechanism to display individual delimiters of a given delimiter set. It also assists in maintaining the same type and size across blocks. Being able to display individual delimiters of defined delimiter sets offers a broad range of (ab)use.

Individual delimiters of a set can be selected and displayed by specifying appropriate flags, for example:

$$\texttt{\textbackslash brk(} \to (, \qquad \texttt{\textbackslash brk)} \to)$$

These flags can be combined with the size specifier or other bracket modifiers, e.g.:

$$\texttt{\textbackslash brk\^{}3(} \to \Big(, \qquad \texttt{\textbackslash brk[s]\^{}2)} \to \Big]$$

This enables to conveniently split delimited expressions over blocks of code, e.g.:

$$\texttt{\textbackslash brk[s]\^{}3(\textbackslash frac\{A\}\{B\}} \ldots /\!/ \ldots \texttt{\textbackslash frac\{C\}\{D\}\textbackslash brk[s]\^{}3)} \to \left[\frac{A}{B} \cdots /\!/ \cdots \frac{C}{D}\right]$$

Here, the type and modifiers specifying the delimiter set are repeated for all individual delimiters. To simplify such expressions and their maintenance, the package provides global registers which store the type and size of delimiters. With those, the above expression can be abbreviated as:

$$\texttt{\textbackslash brk[s]\^{}3[} \ldots /\!/ \ldots \texttt{\textbackslash usedelim]} \quad \text{or} \quad \texttt{\textbackslash brk[s]\^{}3[} \ldots /\!/ \ldots \texttt{\textbackslash brk]}$$

Here, `\usedelim]`, alternatively `\brk]`, closes the delimiter set with the type and options previously specified by `\brk...[`. This mechanism respects nesting, and for even broader control there exists a more elaborate solution using registers, see below.

1.4 Philosophy

Semantic typesetting is one of the philosophies behind LaTeX: The author should focus on the content while the layout is taken care of by the engine. The (body of a) source file largely codes the contents of the document, while the layout is largely specified by the kernel, classes, styles and macro definitions (in the preamble). In order for the separation of content and layout to work well, the meaning of the content must be accurately specified by the source file so that the appropriate layout can be applied to it. For example, a left bracket '`(`' can have many meanings, which the engine could not possibly guess. Even a simple compound expression such as `[A,B]` could have different meanings depending on context such as a compact interval or a commutator. A semantic coding of the latter two concepts such as `\intv{A}{B}` vs. `\comm{A}{B}` clearly distinguishes between them. This allows the typesetting engine to represent them appropriately in every situation. It also allows to consistently define or adjust the typeset representation globally according to one's taste, such as $[A, B]$ vs. $[A; B]$. The price to pay is a larger number of abstract commands (which possibly evaluate to the same expression) and using them to specify the semantics throughout the source file (or at least where practical and useful). Conversely, the price to pay for an immediate typesetting scheme is that all notations need to be fixed at the start, and later adjustments require an elaborate search and replacement of (somewhat ambiguous) patterns like `[`x`,`y`]`.

Another distinction between TeX and LaTeX is that the former frequently uses free-format expressions such as `{x\over y}` whereas the latter normally uses structured commands with arguments such as `\frac{x}{y}`. In that sense, the construct `\left(ax+b\right)` belongs to the world of TeX, whereas an expression like `\delim()*{ax+b}` fits the LaTeX framework better.

1.5 Related CTAN Packages

There are at least three other packages which offer a similar functionality:

- The package delim supplies a command `\delimdef` to declare a set of delimiters which is similar to the present `\DeclareMathDelimiterSet`. The size of delimiters to be used in each case is then specified by a prefix command such as `\mbig` or `\mauto`.
- The package mathtools supplies commands `\DeclarePairedDelimiter...` (among many other things) which are similar to the present `\DeclareMathDelimiterSet`. The size of delimiters to be used in each case is then specified by an optional argument such as '`*`' or `[\big]`.
- The package delimseasy defines a collection of useful delimiters such as `\prn` for round parentheses or `\sqpr` square parentheses. Modifier letters can be prepended and appended to adjust their size.

A functionality of the present package not offered by any of the above packages is to typeset delimiters on the fly, e.g.:

$$\texttt{\textbackslash delim<|>!\{\textbackslash psi\}\{\textbackslash psi\}} \rightarrow \langle\psi|\psi\rangle$$

The mechanism to specify the size is leaner in the sense that it uses only a single character and a single command.

2 Usage

To use the package delimset add the command

```
\usepackage{delimset}
```

to the preamble of your LaTeX document. If not yet present, the package amsmath will be loaded automatically.

2.1 Delimiter Sets and Presentation Flags

The package provides commands to represent various delimiter sets, see section 2.2 and section 2.3 below. Their display can be modified systematically by specifying flags. A delimiter set *name*, e.g. `\brk`, is typically invoked by a sequence like:

$$\backslash\textit{name}\ \textit{flags}\ \texttt{\{}\textit{expr1}\texttt{\}\{}\textit{expr2}\texttt{\}}\ldots$$

where *expr(n)* are the terms to be enclosed by the delimiters and the optional *flags* adjust the presentation of delimiters. The *flags* are described as follows:

First, a size modifier flag controls the size of the delimiters to be displayed. It can take one of the following values:

`0`	default size (0.96 times default line height)
`!`, `+`, `^1`	size `\big` (1.2 times default line height)
`^2`	size `\Big` (1.8 times default line height)
`^3`	size `\bigg` (2.4 times default line height)
`^4`	size `\Bigg` (3.0 times default line height)
`^`n, `^{`x`}`	flexible size adjustment (n=`0`...`9`; x: decimal number)
`_{`h`}`	adjustment to absolute height h (centred)
`*`	variable size `\left...\right`

The intended delimiter size is stored as the dimension variable:

`\delimsize`

It may be used to adjust spacing within the delimiter set proportional to the size of the delimiters. Note that the actual size of delimiters may deviate depending on availability within the given font. Note further that the size cannot be determined for variable-size delimiters (‘`*`’) and thus size `\Big` is assumed for this case. Note finally that the sub/superscript styles are not taken into account and `\delimsize` applies to the default case `\textstyle`. The package also supplies a macro to add horizontal glue proportional to the intended delimiter size:

`\kerndelim{`*width*`}`

Here, *width* is the amount of horizontal glue in units of mu scaled by the ratio of `\delimsize` and the current math font height. The macro does adjust for the selected script style.

Second, a delimiter selection flag is used to pick out and display just one individual delimiter within the set. It can take one of the following values:

`(`	left delimiter
`|`, `,`	first intermediate delimiter
`?`*n*	*n*-th delimiter (starting with `0`: left)
`)`	right delimiter

Third, register flags store and retrieve delimiter sets for splitting them across blocks or lines:

`[`	push delimiter type to register stack, display left delimiter (similar to ‘`>.(`’)
`]`	pop delimiter type from stack, display right delimiter (similar to ‘`<.)`’)
`>`*r*, `>{`*reg*`}`	store delimiter type to register *r* or *reg*
`<`*r*, `<{`*reg*`}`	retrieve delimiter type from register *r* or *reg*

A register stores the type of delimiter set (`\`*name*), the desired size (except for variable size ‘`*`’) as well as the math class of the compound (‘`’`’ vs. ‘`"`’, see below). The register stack enables convenient access to nested delimiter sets, and the special register ‘`.`’ points to the top of the stack. A stored register can be retrieved and displayed by the macro:

`\usedelim` *r flags sel* or `\usedelim{`*reg*`}` *flags sel* or `\usedelim` *sel*

Here, *sel* is a flag to select an individual delimiter (‘`(`’, ‘`|`’, ‘`,`’, ‘`?`*n*’, ‘`)`’ or ‘`]`’), see above. The abbreviated form without register specification operates on the top of the register stack.

In addition, there are some general purpose flags:

`.`	terminates flag processing
`:`	enclose delimiter set in block (for using total box dimensions)
`'`	render delimiter set as `\mathopen` and `\mathclose`
`"`	render delimiter set as `\mathinner`
`-`	display phantom delimiters (for individual delimiters)

Please note the following:

- A delimiter selector (including the combined flags ‘`[`’ and ‘`]`’) terminates flag processing, therefore all relevant flags must be specified before it.
- Parsing also stops with the opening of the first term argument by means of the opening brace ‘`{`’.

- Parsing can also be terminated manually by specifying the flag '`.`' (in case the opening brace '`{`' for arguments is missing for whatever reason).
- Any other letter or token at the location of *flags* is interpreted as an unrecognised flag and triggers a compiler error. For instance, `\brk{x}` must not be abbreviated as `\brk x` because '`x`' would be interpreted as a flag (it might be abbreviated as `\brk.x`).
- The legacy size modifier flags '`1`' ... '`4`' provided for backward compatibility have the same effect as '`^1`'...'`^4`'.
- The phantom delimiter flag '`-`' does not work for variable size. It is intended for reserving space for individual delimiters when they are composed manually into sets (e.g. across lines).
- Technically, individual delimiters can be produced in the variable size version (even though it may defeat the purpose of the delimiter selection mechanism). One has to pay attention when using only the opening or only the closing delimiters from the package and counterbalancing it by direct code for the opposite side. This is because '`*(`' and '`*)`' do not directly translate to `\left`*delim* and `\right`*delim* due to proper spacing, see section 1.2. Instead, they translate to:

  `*(` → `\mathopen{}\mathclose\bgroup\left`*delim*
  `*)` → `\right`*delim*`\egroup`

  One should use analogous code to complete the manual variable size block.
  Note that similar restrictions apply to using the flag '`:`' for block enclosure which uses `\bgroup` and `\egroup` as well.
  As it does not make much sense to use variable-size ('`*`') and block ('`:`') compounds for individual delimiter selection, the state of these flags is not stored in the global registers.
- The delimiter selectors '`[`' and '`]`' implement a stack, and they must be properly balanced. Technically, they access the register '`.`*n*' with the stack counter *n*=1,2,3... a global variable which is increased or decreased, respectively. The special register '`.`' points to the current top of the stack to enable access to intermediate delimiters.

2.2 Inline Declarations

The package provides three general purpose commands to compose delimiter sets with one, two or three encapsulated expressions:

`\delim{`*l*`}{`*r*`}`*flags*`{`*expr*`}`
`\delimpair{`*l*`}{`*m*`}{`*r*`}`*flags*`{`*expr1*`}{`*expr2*`}`
`\delimtriple{`*l*`}{`*m*`}{`*n*`}{`*r*`}`*flags*`{`*expr1*`}{`*expr2*`}{`*expr3*`}`

The expression(s) *expr(n)* will be surrounded by the delimiters *l* and *r* and, in the case of more than one expression, they will be separated by the delimiters *m*, *n*:

l expr r, *l expr1 m expr2 r*, *l expr1 m expr2 n expr3 r*

Here, *l*, *r*, *m*, *n* should be math delimiters *delim* (elementary symbols which can be used for `\left` and `\right`) or the dot '`.`' for the null delimiter. The delimiter set expression can be adjusted in two ways:

First, size adjustment for the delimiters *l*, *r*, *m*, *n* can be suppressed by a starred variant:

`{*`*token*`}` or `{*{`*expr*`}}`

This allows to use an arbitrary *token* or a compound expression *expr* in place of a delimiter *delim*. This is useful when a delimiter character should not adjust in size (e.g. commas), or if a flexible-size version of the character is not available. One might also use a compound expression to create a null delimiter with non-zero width (the width may be specified by `\kerndelim` to make it proportional to the delimiter size).

Second, the intended math class of the intermediate delimiters *m*, *n* may be adjusted in order to achieve a more appropriate automatic spacing by using `{[`*class*`]`*delim*`}`. Here, *class* specifies the intended math class of the delimiter *delim* according to:

delim, `{`*delim*`}`	no glue for size-adjusted delimiters (same as `[o]`) determined by expression for unadjusted size (`*` variant)
`{[o]`*delim*`}`	no glue, similar to `\big`, `\mathord`, `\mathopen`, `\mathclose`
`{[p]`*delim*`}`	followed by `\thinmuskip`, similar to `\bigp`, `\mathpunct`
`{[i]`*delim*`}`	surrounded by `\thinmuskip`, similar to `\bigi`, `\mathinner`
`{[b]`*delim*`}`	surrounded by `\medmuskip`, similar to `\bigb`, `\mathbin`
`{[m]`*delim*`}`	surrounded by `\thickmuskip`, similar to `\bigm`, `\mathrel`

Please note the following:

- The left and right delimiters *l* and *r* are assigned to special classes: They leave no initial and final glue on the enclosed expressions (same as `\mathopen` and `\mathclose`). From the outside, the delimiter set appears either as a combination of `\mathopen`+`\mathclose` (flag '`'`') or as `\mathinner` (flag '`"`').
- The legacy class `[.]` provided for backward compatibility specifies delimiters with unadjusted size: the compound `{[.]`*expr*`}` has the same effect as `{*{`*expr*`}}`.
- The legacy class `[c]` provided for backward compatibility has the same effect as `[o]`.
- Unadjusted-size delimiters can be combined with math classes as:

  `{[`*class*`]*`*token*`}` or `{[`*class*`]*{`*expr*`}}`

- The opening square bracket '`[`' takes the dual role of indicating the optional `[`*class*`]` argument. To avoid conflicts on intermediate delimiters, opening square bracket delimiters should be encoded as '`{{[]}}`' (for `\delim...`) or '`{[]}`' (for `\selectdelim`) rather than '`[`'.

2.3 Declarations

The above constructs can be used to define new delimiter commands via:

`\newcommand{\`*name*`}{\delim{`*l*`}{`*r*`}`*flags*`}`

`\newcommand{\`*name*`}{\delimpair{`*l*`}{`*m*`}{`*r*`}`*flags*`}`

`\newcommand{\`*name*`}{\delimtriple{`*l*`}{`*m*`}{`*n*`}{`*r*`}`*flags*`}`

Here it makes sense to drop all arguments starting at the optional *flags* argument from the definition. The TEX parsing mechanism will then automatically use the tokens following `\`*name* including the optional size modifier. If any of the encapsulated expression(s) are to be passed as explicit arguments to `\`*name*, one will have to find an alternative way to pass *flags*. However, it is possible to predefine *flags* which can be overridden by further *flags* specified by the user.

The above declarations via `\delim...` should be sufficient for most situations. However, there is an even more flexible way to declare delimiter sets:

`\DeclareMathDelimiterSet{\`*name*`}[`*narg*`]{`*compositor*`}`

The syntax of this command is equivalent to the one of `\newcommand`. The difference is that the command `\`*name* first looks for the *flags* argument as described above in section 2.2. It remembers the desired size for evaluating the macro expression *compositor*. Then it parses the arguments as if the command was declared by `\newcommand`.

As usual, the macro expression *compositor* contains the command arguments specified by `#1`, `#2`, ... Note that these should be encapsulated in groups `{#1}`, `{#2}`, ..., in order to prevent them from overwriting definitions at the level of the current group. It should also contain the desired math delimiters specified by:

`\selectdeliml`[`*`]{*delim*}
`\selectdelim`[*class*][`*`]{*delim*}
`\selectdelimr`[`*`]{*delim*}

The three commands must be in proper sequence starting with `\selectdeliml` followed by arbitrarily many `\selectdelim` and terminated by `\selectdelimr`. The math classes *class* and starred variants are defined analogously to section 2.2. Here, the sizes are adjusted automatically according to the previously specified modifier.

Picking out individual delimiters from a set declared by `\DeclareMathDelimiterSet` cannot be achieved automatically. Instead, a selector method may be specified manually (where needed) by:

```
\DeclareMathDelimiterSel{\name}
  {\selectdeliml{l}[\or\selectdelim{m}...]}
  {\selectdelimr{r}}
```

As many intermediate delimiters as needed can be specified at the end of the first argument; each one must start with '`\or`'.

For example, one might declare a set of double square brackets [[∗]] using:

```
\DeclareMathDelimiterSet{\dsb}[1]
  {\selectdeliml{[}\kerndelim{-1.75}\selectdelim[o]{[}{#1}
  \selectdelim[o]{]}\kerndelim{-1.75}\selectdelimr{]}}
```

Here, the command `\kerndelim{-1.75}` reduces the spacing between the two square brackets by approximately 1.75 mu in default size. This could be used for:

`\dsb^4{\dsb^3{\dsb^2{\dsb^1{\dsb{*}}}}}` → $\Big[\!\Big[\Big[\!\Big[\Big[\!\Big[[[[*]]]]\Big]\!\Big]\Big]\!\Big]\Big]$, `\dsb*...` → $\Big[\!\Big[\frac{x}{y}\Big]\!\Big]$

An appropriate selector method for displaying individual delimiters reads:

```
\DeclareMathDelimiterSel{\dsb}
  {\selectdeliml{[}\kerndelim{-1.75}\selectdelim[o]{[}}
  {\selectdelim[o]]\kerndelim{-1.75}\selectdelimr{]}}
```

Finally, towards a very flexible manual composition of delimiter sets, there is the macro:

`\parsedelimflags{`*compositor*`}{`*selector*`}`*flags*

It parses the modifier flags as described in section 2.1. The argument *compositor* is code that displays the composited delimiter set whereas *selector* is a macro that displays the delimiter number `\selecteddelim` of the set. The *flags* follow immediately and should be terminated by an opening brace '`{`', the terminating flag '`.`' or a delimiter selection flag.

2.4 Default Declarations

The package predefines four commonly used sets of delimiters:

- `\brk[`*type*`]`*flags*`{`*expr*`}` represents a standard bracket around a single expression *expr*. The type of bracket can be specified by the optional argument *type*:

  empty or `[r]`: round (x), `[s]`: square $[x]$, `[c]`: curly $\{x\}$, `[a]`: angle $\langle x \rangle$

- `\eval[`*type*`]`*flags*`{`*expr*`}` represents evaluation of a functional expression *expr*. The type of bracket can be specified by the optional argument *type*:

  empty or `[v]`: $f(x)|_a$, `[s]`: $[f(x)]_a^b$

- `\abs`*flags*`{`*expr*`}` represents the absolute value $|expr|$.
- `\norm`*flags*`{`*expr*`}` represents the norm $\|expr\|$.

The above definitions can be suppressed by setting the package option `stddef` to `false`, see section 2.6. The package also defines some extended sets of delimiters as follows:

- `\pair`*flags*`{`*expr1*`}{`*expr2*`}` represents a pair(ing) $(expr1, expr2)$.
- `\set`*flags*`{`*expr*`}` represents the set $\{expr\}$.
- `\setcond`*flags*`{`*expr*`}{`*cond*`}` represents a set with condition $\{expr|cond\}$.
- `\intv[`*type*`]`*flags*`{`*expr1*`}{`*expr2*`}` represents an interval from *expr1* to *expr2*. The in/exclusion of the bounds can be specified by the optional argument *type*:

  empty or `[c]`: closed $[a, b]$, `[o]`: open $]a, b[$, `[l]`: left-open $]a, b]$, `[r]`: right-open $[a, b[$

- `\avg`*flags*`{`*expr*`}` represents some average $\langle expr \rangle$.
- `\corr`*flags*`{`*expr*`}` represents some correlator $\langle expr \rangle$.
- `\comm`*flags*`{`*expr1*`}{`*expr2*`}` represents the commutator $[expr1, expr2]$.
- `\acomm`*flags*`{`*expr1*`}{`*expr2*`}` represents the anti-commutator $\{expr1, expr2\}$.
- `\bra`*flags*`{`*expr*`}` represents a bra-vector $\langle expr|$ in quantum mechanics.
- `\ket`*flags*`{`*expr*`}` represents a ket-vector $|expr\rangle$ in quantum mechanics.
- `\braket`*flags*`{`*expr1*`}{`*expr2*`}` represents a bra-ket contraction $\langle expr1|expr2\rangle$.
- `\lfrac[`*type*`]`*flags*`{`*expr1*`}{`*expr2*`}` describes the linear representation *expr1*/*expr2* of a fraction. An optional argument *type* taking values `[r]`, `[s]`, `[c]`, `[a]` encloses the fraction in round, square, curly or angle brackets as for `\brk`.

The extended definitions need to be activated by the package option `extdef`, see section 2.6.

If the representations of the above delimiters do not suit the purpose or taste of the user, they can be redefined with `\renewcommand`.

2.5 Auxiliary Commands

In addition to the `\bigl`, `\bigr` and `\bigm` commands (as well as their `\Big.`, `\bigg.` and `\Bigg.` counterparts), the package defines three additional sets `\bigp`, `\bigb` and `\bigi` (and counterparts). Here `\bigp` implies the math class `\mathpunct`, `\bigb` the class `\mathbin` and `\bigi` the class `\mathinner`.

Furthermore, the package overloads the size calculation in the `\big...` commands to properly account for the math styles in sub/superscripts (`\scriptstyle`) and nested sub/superscripts (`\scriptscriptstyle`). The latter behaviour can be controlled by the package option `scriptstyle`, see section 2.6.

2.6 Package Options

Options can be passed to the package by:

`\usepackage[`*opts*`]{delimset}` or `\PassOptionsToPackage{`*opts*`}{delimset}`

Here *opts* is a comma-separated list of the available options:

- `stddef`[`=true`|`false`] controls the activation of standard delimiter definitions specified in section 2.4. If no value is given `true` is assumed; initially set to `true`.
- `extdef`[`=true`|`false`] controls the activation of extended delimiter definitions specified in section 2.4. If no value is given `true` is assumed; initially set to `false`.
- `scriptstyle`[`=true`|`false`] controls the overwriting of size modifiers explained in section 2.5. If no value is given `true` is assumed; initially set to `true`.

3 Information

3.1 Copyright

This work consists of the files `README.txt`, `delimset.ins` and `delimset.dtx` as well as the derived files `delimset.sty`, `dlmssamp.tex` and `delimset.pdf`.

3.2 Files and Installation

The package consists of the files

`README.txt`	readme file
`delimset.ins`	installation file
`delimset.dtx`	source file
`delimset.sty`	package file
`dlmssamp.tex`	sample file
`delimset.pdf`	manual

The distribution consists of the files `README.txt`, `delimset.ins` and `delimset.dtx`.

- Run (pdf)LaTeX on `delimset.dtx` to compile the manual `delimset.pdf` (this file).
- Run LaTeX on `delimset.ins` to create the package `delimset.sty` and the sample `dlmssamp.tex`. Copy the file `delimset.sty` to an appropriate directory of your LaTeX distribution, e.g. *texmf-root*`/tex/latex/delimset`.

3.3 Interaction with CTAN Packages

The package is related to other packages available at CTAN:

- Compatibility with the amsmath package has been tested with v2.15d (2016/06/28) and v2.17t (2024/11/05).
- This package uses the package keyval from the graphics bundle to process optional arguments to the package options. Compatibility with the keyval package has been tested with v1.15 (2014/10/28).
- The package icomma modifies the spacing behaviour of the comma character in math mode which leads to inadequate spacing when it is used as a middle delimiter. To achieve proper spacing in sample code, use `{[p]*,}` rather than `{*,}`. The extended commands `\intv`, `\comm`, `\acomm` and `\pair` are compatible with the icomma package has been tested with v2.0+ (2002/03/10).

3.4 Revision History

v2.2.1: 2025/03/25

- maintenance and manual update

v2.2: 2025/02/27

- replicate the amsmath mechanism for tracking the vertical height of a default delimiter in order to be independent of amsmath

v2.1: 2025/01/05

- the flags '`[`' and '`]`' now implement a register stack for convenient individual access to nested delimiter sets
- syntax of `\usedelim` extended for direct access to top of register stack

v2.0: 2024/07/17

- option to display individual delimiters and to store them in global registers across blocks (columns, lines, etc.)
- more general handling of unadjusted-size delimiters
- more general size adjustments
- added inner math class for intermediate delimiters
- added flag to enclose by open/close and inner math class
- added flag to display phantom delimiters
- selected size accessible by `\delimsize` and add proportional kerning by `\kerndelim`.
- added extended definition `\lfrac` for plain inline fractions
- null delimiter now properly has zero width for variable size
- internal mechanisms revised
- compatibility with icomma package (thanks to Olivier Godin for pointing out the issue)

v1.1: 2018/12/30

- classes added, class and size selection mechanism simplified

v1.01: 2018/01/17

- manual rearranged

v1.0: 2016/11/01

- extended standard definitions
- manual and installation package added
- first version published on CTAN

v0.5–0.7: 2016/05/08 – 2016/09/04

- basic functionality
- standard definitions

A Sample File

In this section we provide a LaTeX example how to use some of the delimset features. Preamble and beginning of document body:

```
\documentclass[12pt]{article}

\usepackage[margin=2cm]{geometry}
\usepackage{amsfonts}
\usepackage{delimset}

\begin{document}
```

sizes for default brackets:

```
\[
\brk^0{x},\quad
\brk^1{x},\quad
\brk^2{x},\quad
\brk^3{x},\quad
\brk^4{x}
\]
```

styles for default brackets:

```
\[
\brk[r]{x},\quad
\brk[s]{x},\quad
\brk[c]{x},\quad
\brk[a]{x}
\]
```

nested brackets:

```
\[
\brk[c]^2{\brk[s]!{\brk{ax+b}x+c}x+d}
\]
```

default absolute value, norm and default evaluations:

```
\[
\abs*{\frac{ax+b}{cx+d}},\qquad
\norm*{\frac{ax+b}{cx+d}},\qquad
\eval*{\frac{ax+b}{cx+d}}_{x=0},\qquad
\eval[s]*{\frac{ax+b}{cx+d}}_{x=0}^{x=\infty}
\]
```

outer delimiter spacing:

```
\[\begin{array}{ll}
\square\brk^0{x}\square,&\square\brk^1{A^k}\square,
\\
\square\brk*{x}\square,&\square\brk*{A^k}\square
\end{array}\]
```

delimiter sizes in exponents:

```
\[
e^{\brk{ax+b}},\qquad
e^{\brk!{ax+b}}
\]
```

delimiter declaration:

```
\DeclareMathDelimiterSet{\braket}[2]
  {\selectdeliml<#1\selectdelim|#2\selectdelimr>}
\[
\braket!{\psi}{\psi},
\quad
\braket*{\psi}{\psi\big.}
\]
```

delimiter usage:

```
\[
\delimpair<|>!{\psi}{\psi}
\]
```

conditional set, alternative layouts:

```
\[
\delimpair\{{[m]|}\}!{2n}{n\in\mathbb{Z}},
\quad
\delimpair\{{[b]|}\}!{2n}{n\in\mathbb{Z}},
\quad
\delimpair\{{[i]|}\}!{2n}{n\in\mathbb{Z}},
\quad
\delimpair\{{[p]|}\}!{2n}{n\in\mathbb{Z}},
\quad
\delimpair\{|\}!{2n}{n\in\mathbb{Z}},
\quad
\delimpair\{{*;}\}!{2n}{n\in\mathbb{Z}}
\]
conditional set, alternative layouts with variable size:
\[
\delimpair\{{[m]|}\}*{2n}{n\in\mathbb{Z}\big.},
\quad
\delimpair\{{[b]|}\}*{2n}{n\in\mathbb{Z}\big.},
\quad
\delimpair\{{[i]|}\}*{2n}{n\in\mathbb{Z}\big.},
```

```
\quad
\delimpair\{{[p]|}\}*{2n}{n\in\mathbb{Z}\big.},
\quad
\delimpair\{|\}*{2n}{n\in\mathbb{Z}\big.},
\quad
\delimpair\{{*;}\}*{2n}{n\in\mathbb{Z}\big.}
\]
```

delimiter definition:

```
\newcommand{\comm}{\delimpair[{*,}]}
\[
\comm!{\comm{A}{B}}{C}
+\comm!{\comm{B}{C}}{A}
+\comm!{\comm{C}{A}}{B}
=0
\]
```

alternative representation:

```
\renewcommand{\comm}{\delimpair[{*;}]}
\[
\comm!{\comm{A}{B}}{C}
+\comm!{\comm{B}{C}}{A}
+\comm!{\comm{C}{A}}{B}
=0
\]
```

display individual delimiters of a set with nesting:

```
\renewcommand{\braket}{\delimpair<|>}
\[
\braket{A}{B}
\to \braket( A \braket| B \braket),
\quad
\braket*( A\big. \braket*| B_{} \braket*),
\quad
\braket^1( A \braket^3| B \braket^2),
\quad
\braket^2[ \brk[s]^1[ A \brk] \usedelim| B \usedelim]
\]
```

placing indices before a bracket (does not work in variable-size mode because the final size is not available for the enclosed expressions):

```
\DeclareMathDelimiterSet{\quadindex}[5]
  {\selectdeliml.^{#2}_{#3}\mathord{}\selectdelim[o][
  {#1}\selectdelim[o]]^{#4}_{#5}\selectdelimr.}
\[
\quadindex^2{\frac{x}{y}}{1}{2}{3}{4}
\]
```

End of document body:

```
\end{document}
```

B Implementation

In this section we describe the package `delimset.sty`.

Required Packages. The package loads the package keyval if not yet present. keyval is used for extended options processing.

```
\RequirePackage{keyval}
```

Package Options. The package has some boolean keyval options which can be set to `true` or `false`.

```
\newif\ifdlm@std\dlm@stdtrue
\newif\ifdlm@ext\dlm@extfalse
\newif\ifdlm@script\dlm@scripttrue

\def\dlm@group{dlm@}
\define@key{\dlm@group}{stddef}[true]{\csname dlm@std#1\endcsname}
\define@key{\dlm@group}{extdef}[true]{\csname dlm@ext#1\endcsname}
\define@key{\dlm@group}{scriptstyle}[true]{\csname dlm@script#1\endcsname}

\DeclareOption*{\expandafter\setkeys\expandafter\dlm@group%
  \expandafter{\CurrentOption}}
\ProcessOptions
```

Improved Size Adjustments. Overwrite the amsmath command `\bBigg@` to select the size according to the present math style (uses the amsmath definitions `\@mathmeasure` and `\big@size`). This code is activated only if the package option `scriptstyle` is set to `true`.

```
\ifdlm@script
\ifdefined\bBigg@
\def\bBigg@choice#1#2#3#4{%
  {\@mathmeasure\z@{\nulldelimiterspace\z@}%
     {#1\left#4\vcenter to#3\dimexpr#2\big@size\relax{}\right.}%
   \box\z@}}
\def\bBigg@#1#2{\leavevmode@ifvmode{\mathchoice%
  {\bBigg@choice{\displaystyle}{1}{#1}{#2}}%
  {\bBigg@choice{\textstyle}{1}{#1}{#2}}%
  {\bBigg@choice{\scriptstyle}{0.7}{#1}{#2}}%
  {\bBigg@choice{\scriptscriptstyle}{0.5}{#1}{#2}}}}
\fi
\fi
```

Define punctuation marks (`\bigp`, etc.), binary operators (`\bigb`, etc.) and inner class (`\bigi`, etc.).

```
\providecommand{\bigp}{\mathpunct\big}
\providecommand{\Bigp}{\mathpunct\Big}
\providecommand{\biggp}{\mathpunct\bigg}
\providecommand{\Biggp}{\mathpunct\Bigg}
\providecommand{\bigb}{\mathbin\big}
\providecommand{\Bigb}{\mathbin\Big}
\providecommand{\biggb}{\mathbin\bigg}
\providecommand{\Biggb}{\mathbin\Bigg}
\providecommand{\bigi}{\mathinner\big}
\providecommand{\Bigi}{\mathinner\Big}
\providecommand{\biggi}{\mathinner\bigg}
\providecommand{\Biggi}{\mathinner\Bigg}
```

Size Adjustment Definitions. Replicate the amsmath implementation for tracking the vertical size of a default delimiter in the current font. Our `\dlm@size` matches with

`\big@size` in amsmath which is 1.2 times the vertical size of '(':

```
\newlength\dlm@size
```

The macros `\dlm@size@setchar` and `\dlm@size@scan` scan the font and code point in unicode (xetex/luatex) vs. TEX (etex) engines:

```
\ifdefined\Umathcharnumdef
  \def\dlm@size@setchar{\Umathcharnumdef\dlm@size@char\Umathcodenum`\(\relax}
  \def\dlm@size@scan#1"#2"#3"#4\relax{\dlm@size@get{#3}{#4}}
\else
  \def\dlm@size@setchar{\mathchardef\dlm@size@char\mathcode`\(\relax}
  \def\dlm@size@scan#1"#2#3#4\relax{\dlm@size@get{#3}{#4}}
\fi
```

The macro `\dlm@size@get` computes 1.2 times the vertical height of '(':

```
\def\dlm@size@get#1#2{%
  \dlm@size\fontcharht\textfont"#1"#2\relax%
  \advance\dlm@size\fontchardp\textfont"#1"#2\relax%
  \global\dlm@size1.2\dlm@size}
```

The macro `\dlm@size@reset` updates `\dlm@size`:

```
\newcommand{\dlm@size@reset}{%
  \dlm@size@setchar\expandafter\dlm@size@scan\meaning\dlm@size@char\relax}
```

We can hook onto the amsmath mechanism if already present:

```
\ifdefined\big@size
\def\dlm@size{\big@size}
\else
\addto@hook\every@math@size{\dlm@size@reset}
\fi
```

The macro `\dlm@setvar` enables variable size, the macro `\dlm@setsize` sets a fixed size. Delimiter sizes are implented by `\dlm@big@`. The macro `\kerndelim` adds some kerning proportional to the chosen fixed size of the delimiters (the spacing is merely an approximation, and it does not actually scale for flexible size).

```
\newlength\delimsize
\newcommand{\dlm@setvar}{\let\dlm@ifvar\@firstoftwo\delimsize1.5\dlm@size}
\newcommand{\dlm@setsize}[1]{\let\dlm@ifvar\@secondoftwo%
  \delimsize#1\dlm@size\advance\delimsize\dlm@size\delimsize0.5\delimsize}
\newcommand{\dlm@setabssize}[1]{\let\dlm@ifvar\@secondoftwo\delimsize#1\relax}
\newcommand{\dlm@big@}[1]{{\mathchoice%
  {\dlm@big@choice\displaystyle{1}{#1}}%
  {\dlm@big@choice\textstyle{1}{#1}}%
  {\dlm@big@choice\scriptstyle{0.7}{#1}}%
  {\dlm@big@choice\scriptscriptstyle{0.5}{#1}}}}
\newcommand{\dlm@big@choice}[3]{\hbox{$\m@th\nulldelimiterspace\z@%
  #1\left#3\vcenter to#2\delimsize{}\right.$}}
\newcommand{\kerndelim}[1]{{\delimsize#1\delimsize%
  \mkern\muexpr1.2mu*\delimsize/\dlm@size\relax}}
```

Math Classes Processing. Define class selectors for fixed and unadjusted sizes.

```
\newcommand{\dlm@big}[2]{\dlm@plain{#1}{\dlm@big@#2}}
\newcommand{\dlm@plain}[2]{\dlm@class{#1}{\dlm@phantom{#2}}}
\newcommand{\dlm@class}[1]{\csname dlm@class@#1\endcsname}
\let\dlm@class@\@firstofone
```

```
\let\dlm@class@o\mathord
\let\dlm@class@c\mathord
\let\dlm@class@p\mathpunct
\let\dlm@class@i\mathinner
\let\dlm@class@b\mathbin
\let\dlm@class@m\mathrel
```

Define class selector for variable size. The macro `\dlm@var@null` removes the extra space from variable-size null delimiters (`\left.`, `\middle.` and `\right.` apply the value of `\nulldelimiterspace` only at the end of math processing, even across blocks). The macro `\dlm@var@kern` applies kerning unless in the script styles.

```
\newcommand{\dlm@var@null}[1]{\if.#1\kern-\nulldelimiterspace\fi}
\newcommand{\dlm@var@kern}[1]{\nonscript\mkern#1}
\newcommand{\dlm@var}[1]{\csname dlm@var@#1\endcsname}
\newcommand{\dlm@var@}[1]{\dlm@var@null#1\middle#1}
\let\dlm@var@o\dlm@var@
\let\dlm@var@c\dlm@var@
\newcommand{\dlm@var@p}[1]{\dlm@var@#1\dlm@var@kern\thinmuskip}
\newcommand{\dlm@var@i}[1]{\dlm@var@kern\thinmuskip%
  \dlm@var@#1\dlm@var@kern\thinmuskip}
\newcommand{\dlm@var@b}[1]{\dlm@var@kern\medmuskip%
  \dlm@var@#1\dlm@var@kern\medmuskip}
\newcommand{\dlm@var@m}[1]{\dlm@var@kern\thickmuskip%
  \dlm@var@#1\dlm@var@kern\thickmuskip}
```

Definitions for selecting outer math class `\mathopen`+`\mathclose` vs. `\mathinner`.

```
\newcommand{\dlm@open@i}{\mathinner{}\mathclose{}\mathopen}
\newcommand{\dlm@close@i}{\mathclose{}\mathopen{}\mathinner}
\newcommand{\dlm@inner@oc}{\mathopen{}\mathclose}
\newcommand{\dlm@enclose@openclose}{%
  \let\dlm@open\mathopen\let\dlm@close\mathclose\let\dlm@inner\dlm@inner@oc}
\newcommand{\dlm@enclose@inner}{%
  \let\dlm@open\dlm@open@i\let\dlm@close\dlm@close@i\let\dlm@inner\mathinner}
```

Definitions for selecting inline vs. block insertion.

```
\newcommand{\dlm@inline@l}[1]{\dlm@open{#1}}
\newcommand{\dlm@inline@r}[1]{\dlm@close{#1}}
\newcommand{\dlm@block@l}[1]{\dlm@inner\bgroup\mathopen{#1}}
\newcommand{\dlm@block@r}[1]{\mathclose{#1}\egroup}
\newcommand{\dlm@enclose@inline}{%
  \let\dlm@class@l\dlm@inline@l\let\dlm@class@r\dlm@inline@r}
\newcommand{\dlm@enclose@block}{%
  \let\dlm@class@l\dlm@block@l\let\dlm@class@r\dlm@block@r}
```

Opening and closing definitions.

```
\newcommand{\dlm@plain@l}[1]{\dlm@class@l{\dlm@phantom{#1}}}
\newcommand{\dlm@plain@r}[1]{\dlm@class@r{\dlm@phantom{#1}}}
\newcommand{\dlm@big@l}[1]{\dlm@plain@l{\dlm@big@#1}}
\newcommand{\dlm@big@r}[1]{\dlm@plain@r{\dlm@big@#1}}
\newcommand{\dlm@var@l}[1]{\dlm@inner\bgroup\dlm@var@null#1\left#1}
\newcommand{\dlm@var@r}[1]{\right#1\dlm@var@null#1\egroup}
\newcommand{\dlm@var@pl}[1]{\dlm@var@l.\mathopen{#1}}
\newcommand{\dlm@var@pr}[1]{\mathclose{#1}\dlm@var@r.}
```

Delimiter Storage. Macros for storing and retrieving delimiter types using global registers.

```
\newcount\dlm@reg@lvl
\dlm@reg@lvl\z@
\def\dlm@reg@dot{.}
\newcommand{\dlm@reg@set}[1]{\edef\dlm@reg@cur{#1}\ifx\dlm@reg@cur\dlm@reg@dot%
  \edef\dlm@reg@cur{\dlm@reg@dot\the\dlm@reg@lvl}\fi}
\newcommand{\dlm@reg@step}[1]{\global\advance\dlm@reg@lvl#1%
  \ifnum\dlm@reg@lvl<\z@\PackageError{delimset}{register stack exhausted}{}%
  \dlm@reg@lvl\z@\fi}
\newcommand{\dlm@reg@save@init}[1]{%
  \xdef#1{\delimsize\the\delimsize}\ifx\dlm@inner\mathinner%
  \expandafter\gdef\expandafter#1\expandafter{#1\dlm@enclose@inner}\fi}
\newcommand{\dlm@reg@save}[2]{\dlm@reg@set{#1}%
  \expandafter\dlm@reg@save@init\csname dlm@reg@init@\dlm@reg@cur\endcsname%
  \expandafter\gdef\csname dlm@reg@sel@\dlm@reg@cur\endcsname{#2}}
\newcommand{\dlm@reg@init}[1]{\dlm@reg@set{#1}%
  \csname dlm@reg@init@\dlm@reg@cur\endcsname}
\newcommand{\dlm@reg@sel}{\csname dlm@reg@sel@\dlm@reg@cur\endcsname}
```

Flags Processing. The macro `\parsedelimflags` parses the optional argument(s) following `\delim...` commands to adjust the presentation. Parsing is terminated if the next character in line begins a group (‘`{`’) and the delimiter set is composed by executing argument `#1`. A size flag is stored and parsing continues. A delimiter selector flag immediately displays the desired delimiter by executing argument `#2`. Unknown flags produce an error message.

```
\newcommand{\parsedelimflags}{\dlm@setsize{0.6}\let\dlm@phantom\@firstofone%
  \dlm@enclose@openclose\dlm@enclose@inline\dlm@parseflags}
\newcommand{\dlm@parseflags}[2]{%
  \@ifnextchar\bgroup{#1}{\dlm@parseflag{#1}{#2}}}
\newcommand{\dlm@parseflag}[3]{\begingroup%
  \ifcsname dlm@parseflag@\string#3\endcsname%
  \def\dlm@do{\csname dlm@parseflag@\string#3\endcsname{#1}{#2}}\else%
  \def\dlm@do{\PackageError{delimset}%
  {unknown delimiter set flag ‘\string#3’}{}#1}%
  \fi\expandafter\endgroup\dlm@do}
\newcommand{\dlm@parsedef}[3]{%
  \expandafter\def\csname dlm@parseflag@\string#1\endcsname ##1##2#2{#3}}
```

Note that the delimited expression should be contained within a group such that nested delimiters will not overwrite the outer size definition.

Flags to adjust size. Note that sizes 0 and 1 amount to 0.96 and 1.2 times the current empty math box height.

```
\dlm@parsedef{!}{}{\dlm@setsize{1}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{+}{}{\dlm@setsize{1}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{0}{}{\dlm@setsize{0.6}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{1}{}{\dlm@setsize{1}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{2}{}{\dlm@setsize{2}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{3}{}{\dlm@setsize{3}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{4}{}{\dlm@setsize{4}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{^}{#3}{\dlm@setsize{#3}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{_}{#3}{\dlm@setabssize{#3}\dlm@parseflags{#1}{#2}}
\dlm@parsedef{*}{}{\dlm@setvar\dlm@parseflags{#1}{#2}}
```

Flags to select individual delimiters.

```
\dlm@parsedef{(}{}{\def\selecteddelim{0}#2}
\dlm@parsedef{|}{}{\def\selecteddelim{1}#2}
\dlm@parsedef{,}{}{\def\selecteddelim{1}#2}
\dlm@parsedef{)}{}{\def\selecteddelim{9}#2}
\dlm@parsedef{?}{#3}{\def\selecteddelim{#3}#2}
```

Flags to access registers.

```
\dlm@parsedef{[}{}{\dlm@reg@step\@ne\dlm@reg@save\dlm@reg@dot{#2}%
  \def\selecteddelim{0}#2}
\dlm@parsedef{]}{}{\dlm@reg@init\dlm@reg@dot\dlm@reg@step\m@ne%
  \def\selecteddelim{9}\dlm@reg@sel}
\dlm@parsedef{>}{#3}{\dlm@reg@save{#3}{#2}}
\dlm@parsedef{<}{#3}{\dlm@reg@init{#3}\dlm@parseflags%
  {\PackageError{delimset}{must select delimiter}{}}{\dlm@reg@sel}}
```

Further flags.

```
\dlm@parsedef{.}{}{#1}
\dlm@parsedef{:}{}{\dlm@enclose@block\dlm@parseflags{#1}{#2}}
\dlm@parsedef{'}{}{\dlm@enclose@openclose\dlm@parseflags{#1}{#2}}
\dlm@parsedef{"}{}{\dlm@enclose@inner\dlm@parseflags{#1}{#2}}
\dlm@parsedef{-}{}{\let\dlm@phantom\phantom\dlm@parseflags{#1}{#2}}
```

Delimiter Display. The command `\selectdelim` reproduces the delimiter in argument `#2` using the math class given in argument `#1` and the previously stored size. If the class identifier is '`.`', just return the delimiter argument as is. The commands `\selectdeliml` and `\selectdelimr` implement the left and right delimiter classes, respectively.

```
\newcommand{\selectdelim}[1][]{\begingroup\def\dlm@do{\@ifstar{\dlm@plain{#1}}%
  {\dlm@ifvar{\dlm@var{#1}}{\dlm@big{#1}}}}%
  \if.#1\let\dlm@do\@empty\fi\expandafter\endgroup\dlm@do}
\newcommand{\selectdeliml}{\@ifstar%
  {\dlm@ifvar\dlm@var@pl\dlm@plain@l}{\dlm@ifvar\dlm@var@l\dlm@big@l}}
\newcommand{\selectdelimr}{\@ifstar%
  {\dlm@ifvar\dlm@var@pr\dlm@plain@r}{\dlm@ifvar\dlm@var@r\dlm@big@r}}
```

Declaration of New Delimiter Commands. The macro `\DeclareMathDelimiterSet` declares a new set of delimiters as the macro '`\`*name*'. This macro checks for optional flags and stores the desired size. It then passes on to a second macro '`\dlm@dcl@`*name*', which takes the actual code. If the flags select an individual delimiter, the macro '`\dlm@sel@`*name*' is called instead. The latter macro should be defined via `\DeclareMathDelimiterSel` such that it produces the desired delimiter number `\selecteddelim`.

```
\newcommand{\DeclareMathDelimiterSet}[1]{\expandafter\dlm@declare%
  \csname dlm@dcl@\expandafter\@gobble\string#1\endcsname{#1}}
\def\dlm@declare#1#2{\expandafter\dlm@declare@%
  \csname dlm@sel@\expandafter\@gobble\string#2\endcsname{#1}{#2}}
\def\dlm@declare@#1#2#3{\newcommand{#3}{\parsedelimflags{#2}{#1}}%
  \providecommand{#1}{}\newcommand{#2}}
\newcommand{\DeclareMathDelimiterSel}[3]{\expandafter\def%
  \csname dlm@sel@\expandafter\@gobble\string#1\endcsname%
  {\ifcase\selecteddelim#2\else#3\fi}}
```

Inline Delimiter Declarations. Inline declaration for delimiters via `\delim...`. The following code is similar to the one produced by `\DeclareMathDelimiterSet`, but the delimiter arguments are processed *before* the optional size modifier.

`\delim` is used for a single delimited expression.

```
\newcommand{\delim}[2]{%
  \parsedelimflags{\dlm@dcl@delim{#1}{#2}}{\dlm@sel@delim{#1}{#2}}}
\newcommand{\dlm@dcl@delim}[3]{%
  \selectdeliml#1{#3}\selectdelimr#2}
\newcommand{\dlm@sel@delim}[2]{\ifcase\selecteddelim\selectdeliml#1%
  \else\selectdelimr#2\fi}
```

`\delimpair` is used for two delimited expressions separated by an intermediate delimiter.

```
\newcommand{\delimpair}[3]{%
  \parsedelimflags{\dlm@dcl@delimpair{#1}{#2}{#3}}%
  {\dlm@sel@delimpair{#1}{#2}{#3}}}
\newcommand{\dlm@dcl@delimpair}[5]{%
  \selectdeliml#1{#4}\selectdelim#2{#5}\selectdelimr#3}
\newcommand{\dlm@sel@delimpair}[3]{\ifcase\selecteddelim\selectdeliml#1%
  \or\selectdelim#2\else\selectdelimr#3\fi}
```

`\delimtriple` is used for three delimited expressions separated by two intermediate delimiters.

```
\newcommand{\delimtriple}[4]{%
  \parsedelimflags{\dlm@dcl@delimtriple{#1}{#2}{#3}{#4}}%
  {\dlm@sel@delimtriple{#1}{#2}{#3}{#4}}}
\newcommand{\dlm@dcl@delimtriple}[7]{%
  \selectdeliml#1{#5}\selectdelim#2{#6}\selectdelim#3{#7}\selectdelimr#4}
\newcommand{\dlm@sel@delimtriple}[4]{\ifcase\selecteddelim\selectdeliml#1%
  \or\selectdelim#2\or\selectdelim#3\else\selectdelimr#4\fi}
```

`\usedelim` retrieves a stored delimiter type. The arguments '`]`', '`(`', '`|`', '`,`', '`?`n', '`)`' immediately select the delimiter using the stack register. Otherwise the desired register must be followed by the delimiter selection flag.

```
\newcommand{\usedelim}{%
  \@ifnextchar]{\parsedelimflags{}{}}{%
  \@ifnextchar({\parsedelimflags{}{}<.}{%
  \@ifnextchar|{\parsedelimflags{}{}<.}{%
  \@ifnextchar,{\parsedelimflags{}{}<.}{%
  \@ifnextchar){\parsedelimflags{}{}<.}{%
  \@ifnextchar?{\parsedelimflags{}{}<.}{%
  \parsedelimflags{}{}<}}}}}}
```

Standard Definitions. Define some common delimiters (by `providecommand` so as not to overwrite previously existing commands). This code is activated only if the package option `stddef` is set to `true`.

```
\ifdlm@std
\providecommand{\brk}[1][r]{\begingroup\def\dlm@use{\delim()}%
  \if r#1\def\dlm@use{\delim()}\fi%
  \if s#1\def\dlm@use{\delim[]}\fi%
  \if c#1\def\dlm@use{\delim\{\}}\fi%
  \if a#1\def\dlm@use{\delim<>}\fi%
  \expandafter\endgroup\dlm@use}
\ifdefined\rvert
```

```
\providecommand{\eval}[1][v]{\begingroup\def\dlm@use{\delim.\rvert}%
  \if v#1\def\dlm@use{\delim.\rvert}\fi%
  \if s#1\def\dlm@use{\delim[]}\fi%
  \expandafter\endgroup\dlm@use}
\else
\providecommand{\eval}[1][v]{\begingroup\def\dlm@use{\delim.\vert}%
  \if v#1\def\dlm@use{\delim.\vert}\fi%
  \if s#1\def\dlm@use{\delim[]}\fi%
  \expandafter\endgroup\dlm@use}
\fi
\ifdefined\lvert
\providecommand{\abs}{\delim\lvert\rvert}
\else
\providecommand{\abs}{\delim\vert\vert}
\fi
\ifdefined\lVert
\providecommand{\norm}{\delim\lVert\rVert}
\else
\providecommand{\norm}{\delim\Vert\Vert}
\fi
\fi
```

Extended Definitions. Define some extended delimiters. This code is activated only if the package option `extdef` is set to `true`.

```
\ifdlm@ext
\providecommand{\pair}{\delimpair({[p]*,})}
\providecommand{\set}{\delim\{\}}
\providecommand{\setcond}{\delimpair\{|\}}
\providecommand{\intv}[1][c]{\begingroup%
  \def\dlm@use{\delimpair[{[p]*,}]}%
  \if c#1\def\dlm@use{\delimpair[{[p]*,}]}\fi%
  \if l#1\def\dlm@use{\delimpair]{[p]*,}]}\fi%
  \if r#1\def\dlm@use{\delimpair[{[p]*,}[}\fi%
  \if o#1\def\dlm@use{\delimpair]{[p]*,}[}\fi%
  \expandafter\endgroup\dlm@use}
\providecommand{\avg}{\delim<>}
\providecommand{\corr}{\delim<>}
\providecommand{\comm}{\delimpair[{[p]*,}]}
\providecommand{\acomm}{\delimpair\{{[p]*,}\}}
\providecommand{\bra}{\delim<|}
\providecommand{\ket}{\delim|>}
\providecommand{\braket}{\delimpair<|>}
\providecommand{\lfrac}[1][]{\begingroup%
  \def\dlm@use{\delimpair./.}%
  \if r#1\def\dlm@use{\delimpair(/)}\fi%
  \if s#1\def\dlm@use{\delimpair[/]}\fi%
  \if c#1\def\dlm@use{\delimpair\{/\}}\fi%
  \if a#1\def\dlm@use{\delimpair</>}\fi%
  \expandafter\endgroup\dlm@use}
\fi
```