The physymb package[*]

David Zaslavsky
diazona@ellipsix.net

January 2, 2015

Abstract

This package is obsolete. The physymb package contains a bunch of simple macro definitions that may be useful for typesetting physics papers or other things. All the useful macros are implemented by other packages, which you should use directly instead of physymb.

It has come to my attention that other packages provide much the same functionality as physymb, so I am marking the package obsolete. Here are some packages you can use instead of physymb:

physics gives differentials and derivatives, various sorts of paired delimiters including absolute value and vector norm, vector calculus operators including div, grad, and curl, inverse trigonometric functions, Dirac notation and matrix notation

commath gives differentials, derivatives, and various sorts of paired delimiters

braket gives Dirac notation

siunitx gives units and scientific notation

hepnames gives elementary particles

mandi gives inverse trig functions, signum, unit vectors, vector calculus including div, grad, and curl, and the Griffiths script r. mandi also provides elementary unit typesetting, but I consider siunitx to be superior for that purpose. It also provides notation for differentials and derivatives, but the implementations in commath or physics are probably easier to use.

Many macros in physymb are now implemented in terms of these other packages.

I will leave physymb up on CTAN so old documents can still be compiled. However, there will be no further updates to this package (unless someone reports a bug and makes a convincing case that "use other packages" is not an acceptable workaround).

For the rest of this documentation, when there are a bunch of similar macros that I explain together, I've usually only listed one or two in the left margin. In these cases, all the macros are given in the text.

[*]This document corresponds to physymb v0.3, dated 2014/12/19.

1 Options

`physymb` recognizes the following options, in no particular order.

- `arrowvectors` causes vectors (specifically, the `\vec` command) to be rendered with an arrow above the symbol.
- `boldvectors` causes vectors (again, from `\vec`) to be rendered by typesetting the symbol in bold. It's the alternative to `arrowvectors`.
- `braket` pulls in the `braket` package. (It's precisely equivalent to `\usepackage{braket}`, it's just here for convenience.)
- `feynman` pulls in the `feynmp` package. (It's precisely equivalent to `\usepackage{feynmp}`, it's just here for convenience.)
- `overridemandi` is only relevant if you are using `physymb` and the `mandi` package in the same document, and only if you load `physymb` after `mandi`. If you do, specifying this option causes certain macros in physymb to be defined in a way that will override the definitions of the same macros from `mandi`. Otherwise, the definitions of those commands in `physymb` will be skipped, leaving the definitions in `mandi` to be used. The affected macros are `\evalat`, `\curl`, and `\abs`. (If you specify this option when `mandi` has not been loaded, it has no effect, but a warning will be printed in the log.)
- `particle` enables all the particle physics macros.
- `units` pulls in the `siunitx` package and enables the additional unit macros.

2 Macros

2.1 Trigonometry

`\asin` `\acos` The AMS packages only define inverse trigonometric functions using the "arc" syntax, i.e. they actually prefix "arc" to the name (as in $\arcsin x$). Sometimes you'd rather write them with a superscript -1 to save space, so those versions are imported from the `mandi` package. We have the inverse functions `\asin`, `\acos`, `\atan`, `\asec`, `\acsc`, and `\acot`.

`\sech` `\cosh` For some reason, the hyperbolic sine and cosine `\sech` and `\cosh` aren't defined in the AMS packages, but they are defined in `mandi` and those definitions are incorporated here.

`\asinh` `\acosh` Finally, the inverse hyperbolic trig functions written with the superscript -1 are defined just as with the regular inverse trig functions. We have `\asinh`, `\acosh`, `\atanh`, `\asech`, `\acsch`, and `\acoth`, again all imported from `mandi`.

2.2 Sets

There are certain sets of numbers that are semi-frequently referenced in physics. Typically they're used to say something like $n \in \mathbb{Z}$. Of course, a macro like `\intset` is not necessarily much quicker than writing `\mathbb{Z}`, but these macros are intended to have names that relate to their meanings so that you don't have to remember which letter goes to which set.

`\whlset` `\whlset` ($\mathbb{Q}$) denotes the set of whole numbers, which is typically defined to include all integers greater than zero, although there are different contradictory definitions floating around.

`\natset` `\natset` ($\mathbb{N}$) denotes the set of natural numbers, which is typically defined to include all integers greater than or equal to zero. Some people define "natural numbers" to exclude zero.

`\intset` `\intset` ($\mathbb{Z}$) denotes the set of all integers.

`\realset` `\realset` ($\mathbb{R}$) denotes the set of all real numbers.

`\imagset` `\imagset` ($\mathbb{I}$) denotes the set of all imaginary numbers, which is all complex numbers with real part equal to zero. This one is infrequently used.

`\cpxset` `\cpxset` ($\mathbb{C}$) denotes the set of all complex numbers.

2.3 Calculus

Probably the most useful macros in the package are the derivative operators. Since it's so common to write something of the form $\frac{\mathrm{d}y}{\mathrm{d}x}$ or $\frac{\partial y}{\partial x}$, we have two-character macros for each:

`\ud`
- `\ud{`⟨*top*⟩`}{`⟨*bottom*⟩`}` typesets the normal total derivative

`\pd`
- `\pd{`⟨*top*⟩`}{`⟨*bottom*⟩`}` typesets a partial derivative, which is the same thing but with a partial derivative symbol instead of the d

`\udd` `\uddd` `\pdd` `\pddd` There are variants of these that produce higher-order derivatives; you can add an order by adding another `d`, up to a total of three. These are implemented as wrappers around `\od` and `\pd` from the `commath` package. If you need something higher than the third derivative, you're probably best off using the `commath` macros directly, with their optional argument giving the order of the derivative.

`\udc` `\pdc` The macro `\udc` gives you the character that represents a differential. It's typically set in roman type to distinguish it from a variable. `\pdc` is also defined as the partial derivative character for consistency. There are variants of each with exponents (up to 3) built in; again, you get them by adding an extra `d` or two to the name of the command, `\uddc` and `\udddc` and so on.

`\uds` `\pds` If you're using these in an integral, it's common to want a small space before the differential, so there are variants of the preceding commands defined that include this small space for you; they replace the `c` with an `s`. They follow the same pattern of adding additional `d`'s to get exponents. For example:

```
\iint e^{i\vec{k}\cdot\vec{x}}\udds\vec{x}
```

$$\iint e^{i\mathbf{k}\cdot\mathbf{x}}\,\mathrm{d}^2\mathbf{x}$$

2.4 Vector Calculus

`\div` `\grad` `\curl` `\lapl`

`\physymb` defines `\div`, `\grad`, and `\curl`, to represent the divergence, gradient, and curl, in terms of the corresponding macros from `mandi`.

There is also a macro for the Laplacian operator (divergence of a gradient), `\lapl`, again defined in terms of the macro from `mandi`.

2.5 Complex Analysis

`\conj` There is a macro to indicate the conjugate of a number, `\conj{`⟨*number*⟩`}`. It puts a superscript star after the number, as in z^*.

`\realop` `\imagop` The traditional keywords indicating the real and imaginary parts of a complex number are given macros `\realop` and `\imagop`. They typeset Re and Im respectively.

`\real` `\imag` Why the `op`? Well, there are alternate versions that will also put curly braces around the following argument, `\real` and `\imag`. This is the way Re and Im are often used. (I'm open to changing the definitions of these based on feedback.)

```
\real{z}, \imag{z}
```
$\mathrm{Re}\{z\}, \mathrm{Im}\{z\}$

`\abs` The macro `\abs{`⟨*value*⟩`}` surrounds its argument with vertical bars. It is simply imported from `mandi`.

2.6 Linear Algebra

There are several assorted macros for linear algebra keywords and concepts.

`\vec` `\vecvar` Vectors can be written using the macro `\vec{`⟨*label*⟩`}`, which typesets the ⟨*label*⟩ either in bold or with an arrow over it, according to which option was passed to the package (`arrowvectors` or `boldvectors`). The default is to use an arrow, to resemble the builtin definition of `\vec` (which, by the way, is overridden by this package). In many cases I prefer bold. `\vecvar{`⟨*label*⟩`}` is another macro that does the exact same thing, for consistency with the other kinds of variables.

`\tnsvar` The macro `\tnsvar{`⟨*label*⟩`}` is for typesetting tensors. This just makes the ⟨*label*⟩ bold, it doesn't do anything with indices. If you want a way to typeset tensor indices, look at the tensor package.

`\matvar` `\matvar{`⟨*label*⟩`}` is intended to designate matrices. It makes the label bold.

`\identitym` The macro `\identitym` represents the identity matrix. It typesets a 1 in the same style as `\matvar` (so, bold).

`\determinant` The macro `\determinant{`⟨*matrix*⟩`}` uses vertical bars to denote the determinant of the ⟨*matrix*⟩. It's an alternative to the keyword operator `\det`, which just typesets as det.

`\trace` The macro `\trace` just typesets Tr. It's akin to `\det`.

`\diag` This just typesets diag, which is used to represent a matrix with the given entries on the diagonal. For example, one might write `\diag(1,2,3,4)`.

`\norm` The norm of a vector can be denoted by double vertical bars. This is implemented by `\norm{`⟨*value*⟩`}`.

`\unitx` `\unity` `\unitz` Since it's so common to refer to unit vectors using hat notation, there are a bunch of macros for them using various letters. The package defines `\unitd`,

`\unite`, `\uniti`, `\unitj`, `\unitk`, `\unitl` (which typesets as $\widehat{l}$, not the normal l), `\unitn`, `\unitp`, `\unitq`, `\unitr`, `\units`, `\unitt`, `\unitu`, `\unitv`, `\unitw`, `\unitx`, `\unity`, `\unitz`, and for non-roman characters, `\unitphi`, `\unitrho`, `\unittheta`, and `\unitomega`. If you want to use a different letter as a unit vector, it can be done with `\unitvec{`⟨*symbol*⟩`}`.

`\unitvec`

`\herm` `\herm{`⟨*operator*⟩`}` designates the hermitian conjugate of an operator with a superscript dagger.

`\transpose` `\transpose{`⟨*matrix*⟩`}` sets a superscript T after the matrix to denote the transpose.

`\commut` `\acommut` There are simple macros for the commutator, `\commut{`⟨*operator*⟩`}{`⟨*operator*⟩`}`, and the anticommutator, `\acommut{`⟨*operator*⟩`}{`⟨*operator*⟩`}`. They just put the appropriate kind of braces around the arguments (and the comma between them, of course).

2.7 Differential Geometry

`\exd` The exterior derivative has a macro, `\exd`, kind of like the macro for differentials (d) although typeset in bold to distinguish it. This one doesn't have any variants, though, because $\mathbf{d}^2 = 0$.

`\hodge` The macro `\hodge` just puts a star (not superscript) to represent the Hodge dual. Use it as a prefix to the variable, $\star \mathbf{d}\, x$.

2.8 Classical Mechanics

`\pbrac` The Poisson brackets of a pair of variables can be typeset using the macro `\pbrac{`⟨*function*⟩`}{`⟨*function*⟩`}`. This just surrounds the two arguments with curly braces, producing $\{f, g\}$.

`\pbracvars` If you want to specify which variables the derivatives in the Poisson brackets are being taken with respect to, use the variant

`\pbracvars{`⟨*function*⟩`}{`⟨*function*⟩`}{`⟨*variable*⟩`}{`⟨*variable*⟩`}`

It comes out looking like $\{f, g\}_{q,p}$.

2.9 Quantum Mechanics

If the `braket` option is passed, `physymb` pulls in the `braket` package for writing Dirac notation. See the documentation for that package for details.

Additionally, two semantic macros are provided as alternate names for certain combinations of bras and kets:

`\expect` To get an expectation value (an on-diagonal matrix element with the state left implicit, $\langle A \rangle$), use `\expect{`⟨*operator*⟩`}`. This is just an alternate name for `braket`'s `\braket{`⟨*operator*⟩`}`; the only reason to use it is to make it clear what you meant to someone reading your source code. (A very good reason, as far as I'm concerned) You can put an arbitrary expression within `\expect`, but don't use vertical bars because then it'll look like a matrix element $\langle \psi \mathtt{A} \psi \rangle$.

`\Expect` `\Expect` is the same as `\expect` except that it scales the angle brackets using

`\left` and `\right`.

`\project` For a projection operator (outer product between a state and itself), we have the command `\project{`⟨*label*⟩`}{`⟨*value*⟩`}`, which comes out as $|\psi\rangle\, x \,\langle\psi|$. To get a general outer product between two different states, use `\bra` and `\ket`,

`\ket{`⟨*ket label*⟩`}`⟨*value*⟩`\bra{`⟨*bra label*⟩`}`

`\Project` Again, `\Project` is just like `\project` except that it scales the delimiters.

2.10 Units

If the `units` option is provided to `physymb`, it automatically includes the `siunitx` package and defines some additional units that are often useful in practice. See the documentation of `siunitx` for commands provided by that package.

Additional units The `siunitx` package only includes SI units (as the name would suggest), but there are certain non-SI units that turn out to be occasionally useful when dealing with American non-scientists. `physymb` defines a selection of them as macros.

`\torr` `\mmHg` Torr, `\torr`, and millimeters of mercury, `\mmHg`, are common atmospheric pressure units.

`\amu` `\amu` represents the atomic mass unit, defined as $\frac{1}{12}$ of the mass of a carbon 12 atom.

`\yr` `\yr` represents a year with the symbol yr. There are various definitions of different kinds of years floating around, but generally the symbol is the same.

`\erg` `\erg` represents an erg, the CGS unit of energy, which still finds occasional use. Its value is $1 \times 10^{-7}\,\mathrm{J}$.

`\gauss` `\gauss` is the Gauss, a unit of magnetic field equal to $1 \times 10^{-4}\,\mathrm{T}$.

`\molar` `\molar` represents a molar, a unit of concentration equal to one mole per liter. Strictly speaking, this is a chemistry unit, but it occasionally comes up in physics so it shouldn't hurt to have the macro around.

`\poise` The poise is the CGS unit of viscosity, equal to $0.1\,\mathrm{Pa\,s}$.

`\foot` The foot is the Imperial unit of length, equal to $30.48\,\mathrm{cm}$.

`\mileperhour` This is typically (or perhaps almost exclusively) used to measure transportation speeds: cars, trains, airplanes, etc. It's equal to about $0.447\,\mathrm{m\,s^{-1}}$.

`\pound` `\poundforce` The pound is the Imperial unit of either force or mass, depending on who you ask. Technically I believe it is a force, but in many situations I've often found it clearer to treat it as a unit of mass and use lbf (pound of force) as the unit of force. `physymb` defines macros for both.

In this sense, a pound is equal to about $453.59\,\mathrm{g}$, and the pound of force is the weight of that mass under standard Earth surface gravity, which works out to about $4.448\,\mathrm{N}$.

2.11 Particle Physics

As a particle physicist, I do a lot of work that involves notation for elementary particles, so it's become useful to have a set of macros that produce standard written representations for them. The names of the commands are pretty cryptic, but I've found that once you get used to using them, the names aren't hard to remember and the effort saved by having short macro names at least *feels* worthwhile.

The macros in this package are implemented in terms of the `heppennames` macros. `heppennames` and `hepparticles` will be loaded if the `particles` option is passed to this package.

In general, all the macro names follow the same pattern. Each one ends with a type code that identifies the type of particle: `q` for quark, `lp` for a "regular" lepton, `nu` for a neutrino, `br` for a baryon, `m` for a meson, and `bsn` for a boson. At the beginning is a particle code consisting of one or two letters that identify the specific particle within that type.

Most of the basic macros consist of just those two parts. Antifermion macros are constructed by prepending an `a` to the type code. For vector bosons that occur in charge triplets, you prepend one of `p` (plus), `z` (zero), or `m` (minus) to indicate which one of the triplet you want. The same goes for baryons which occur in "triplets" with the same name (three particles denoted by the same letter, even though they may not actually be a triplet). Singlet baryons have the `z` as well for consistency.

The proton and neutron are named differently because their names are so common.

`\upq` `\dnq` **Quarks** Each of the quark macros is named with three letters. The first two letters are the particle code representing the name of the quark, and the third is the type code `q`. The macros are `\upq`, `\dnq`, `\srq`, `\chq`, `\btq`, and `\tpq`, representing the up, down, strange, charm, bottom, and top quarks, respectively.

`\upaq` `\dnaq` The corresponding macros for the antiquarks are obtained by prepending `a` to the type code `q`. We have `\upaq`, `\dnaq`, `\sraq`, `\chaq`, `\btaq`, and `\tpaq`.

`\elp` `\enu` **Leptons** Leptons are done a little differently because there are two distinct types. The macros for the electron, muon, and tau lepton are named with a letter and `lp`: we have `\elp` for the electron, `\ulp` for the muon, and `\tlp` for the tau. Neutrino macros are constructed using the same first letter, but `nu` instead of `lp`: `\enu`, `\unu`, and `\tnu`.

`\ealp` `\eanu` Antileptons are named with an `a` between the particle code and the type code. So we get `\ealp`, `\ualp`, and `\talp` for the "regular" antileptons and `\eanu`, `\uanu`, and `\tanu` for the antineutrinos.

`\lmzbr` `\sgpbr` `\sgzbr` `\sgmbr` **Baryons** Many of the most commonly referenced baryons in the standard model have macros defined. Each of these ends with the type code `br`. Most of them are built by putting a particle code and a charge letter together: we have `\lmzbr` for the lambda baryon; `\sgpbr`, `\sgzbr`, `\sgmbr` for the sigmas, `\xizbr` and `\ximbr` for the xi particles, and `\ommbr` for the omega of charge -1. The delta macros are named on the same principle but since there are four of them, we use two charge letters to indicate the $+2$ charge: `\dlppbr`, `\dlpbr`, `\dlzbr`, and `\dlmbr`.

`\sgspbr` `\sgszbr` `\sgsmbr` In addition, there are macros for the starred (excited) versions of the sigmas and xis (only), obtained by adding an `s` before the charge letter: `\sgspbr` etc. and `\xiszbr` etc.

`\prbr` `\nebr` The proton and neutron don't quite fall into the pattern because their names aren't used for multiple particles. The proton is `\prbr` and the neutron is `\nebr`.

`\dlmmabr` The antiparticles to all these are obtained in *almost* the usual way, by adding `a` just before the type code `br`. The one difference is that the charge letters are updated to reflect the actual charge of the antiparticle, so for example the antipartcle of the Δ^{++} (`\dlppbr`), the $\overline{\Delta}^{--}$, is written `\dlmmabr`, with two `m`'s because of its double-minus charge.

`\pipm` `\pizm` `\pimm` **Mesons** Essentially all the mesons defined in the standard model have macros. The naming can be a bit tricky because some of them are named as charge triplets while others are named as antiparticles. In the former case, we have the πs, `\pipm`, `\pizm`, and `\pimm`, and the ρs, `\ropm`, `\rozm`, and `\romm`. (I'm not sure if it'd make it cleaner to just add the `h` into the names) The kaons have similar names, `\kapm`, `\kazm`, and `\kamm`, but there is also the $\overline{\mathrm{K}}^0$, `\kazam`. Finally, the neutral mesons are named `\etam`, `\etapm` (here the `p` is for "prime," not "plus"), and `\phim`.

`\phbsn` `\Wpbsn` `\Wmbsn` **Bosons** There aren't that many bosons so the naming is simple: `\phbsn` for the photon, `\Zzbsn` for the neutral Z, and `\Wpbsn` and `\Wmbsn` for the Ws. There's also `\Wbsn`, which does not indicate either charge, for when you need to refer to a generic W boson. The Higgs boson is written `\hbsn`.

`\photon` Also, there is a macro `\photon` which is defined to be the same thing as `\phbsn`. It's included to support some old LaTeX files I wrote and although it will *probably* not be removed from the package in the future, I make no guarantees.

2.12 Miscellaneous

`\scriptr` `\scriptr` produces the script r found in Griffiths' electromagnetism textbook, or at least the closest equivalent in LaTeX, $\mathscr{r}$.

`\orderof` `\orderof{`⟨*expression*⟩`}` represents the order of an expression, for example the error term in a perturbation series. Typical usage would be like

```
\frac{1}{1 - x} = 1 + x + \orderof{x^2}
```

$$\frac{1}{1-x} = 1 + x + \mathcal{O}\left(x^2\right)$$

It can also be used to discuss the growth of a function, e.g. "$\mathcal{O}\left(x^3\right)$ for large x," or for similar uses such as big-O notation in computer algorithm analysis.

`\sgn` There is a macro for the sign operator, `\sgn`, defined as

$$\operatorname{sgn} x = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$$

(and yes, this is not really *complex* analysis)

`\round` Occasionally it's useful to have some way to designate rounding a number. The `\round` macro can be used for that. It comes out as $\operatorname{round}(x)$ (I do recommend the parentheses).

`\evalat` The macro `\evalat{`⟨*expression*⟩`}{`⟨*lower limit*⟩`}{`⟨*upper limit*⟩`}` is mainly useful for when you want to denote the numerical value of a derivative at a specific point, or when you want to represent the evaluation of an integral at the endpoints of the range of integration. It produces a vertical bar at the right of the ⟨*expression*⟩, with the ⟨*lower limit*⟩ and ⟨*upper limit*⟩ typeset at the lower and upper endpoints of the bar, respectively.

`\evalat{x^3 + 3x - 5}{2}{7}` $$\left. x^3 + 3x - 5 \right|_{2}^{7}$$

3 Feedback

This package is always a work in progress, both in terms of adding new macros to the collection and fixing any errors or inconveniences in the ones that are already here. Any feedback you may have will be welcome at my email address, given at the top of the document.

4 Bugs

With certain versions of LaTeX (pdflatex included in TeXLive 2009 comes to mind), there is a register allocation conflict between this package and the `floatrow` and `morefloats` packages. If you're using either of those along with `physymb`, and you're getting `No room for a new \count` errors, you need to add `\usepackage{etex}\reserveinserts{28}` to your preamble. See `http://tex.stackexchange.com/questions/38607/no-room-for-a-new-dimen` for more information.

5 Implementation

5.1 Initialization

```
1 \RequirePackage{ifthen}
```

This flag is set if the `particle` option is enabled. It enables definitions of particle symbol macros.

```
2 \newboolean{pparticle}
```

This flag is set if the `feynman` option is enabled. It pulls in the `feynmf` package.

```
3 \newboolean{pfeynman}
```

This flag is set if the `braket` option is enabled. It pulls in the `braket` package.

```
4 \newboolean{pbraket}
```

This flag is set if the `units` option is enabled. It pulls in the `siunitx` package and provides additional unit definitions.

```
\newboolean{punits}
```

This flag is set if the `boldvectors` option is enabled. It causes vectors to be rendered using a bold font instead of an overset arrow.

```
\newboolean{pboldvectors}
```

This flag is set if the `mandi` option is enabled. It pulls in the `mandi` package.

```
\newboolean{pmandi}
```

5.2 Option Declarations

These are the option declarations, pretty self-explanatory.

```
\DeclareOption{braket}{\setboolean{pbraket}{true}}
\DeclareOption{mandi}{%
 \setboolean{pmandi}{true}%
}
\DeclareOption{particle}{\setboolean{pparticle}{true}}
\DeclareOption{units}{\setboolean{punits}{true}}
\DeclareOption{feynman}{\setboolean{pfeynman}{true}}
\DeclareOption{arrowvectors}{\setboolean{pboldvectors}{false}}
\DeclareOption{boldvectors}{\setboolean{pboldvectors}{true}}
\ProcessOptions\relax
```

This emits a warning that the package is obsolete:

```
\PackageWarning{physymb}{The physymb package is obsolete! See the documentation.}
```

5.3 Macro Definitions

Here we bring in the AMS packages for mathematical notation.

```
\RequirePackage{amsbsy}
\RequirePackage{amsmath}
\RequirePackage{amsfonts}
\RequirePackage{amssymb}
\allowdisplaybreaks[2]
\RequirePackage{accents}
```

Load the `mandi` package if requested

```
\ifthenelse{\boolean{pmandi}}{%
 \RequirePackage{mandi}[2014/12/18]%
}{}
```

Load the `hepparticles` and `heppennames` package if particles are requested

```
\ifthenelse{\boolean{pparticle}}{%
 \RequirePackage{hepparticles}%
 \RequirePackage{heppennames}%
}{}
```

`mandi` is the package that includes the script r, 𝓇.

```
\AtBeginDocument{
 \ifthenelse{\isundefined{\scripty}}{%
  \newcommand{\scriptr}{\PackageError{physymb}{script r requires the mandi package}}%
 }{%
  \newcommand{\scriptr}{\scripty{r}}%
 }
}
```

The `commath` package is used to implement differentials and derivatives.

```
\RequirePackage{commath}
```

Here we load the `braket` package if the corresponding option was passed.

```
\ifthenelse{\boolean{pbraket}}
{
 \RequirePackage{braket}
```

Semantic Dirac notation, implemented on top of braket macros

```
 \newcommand{\project}[2]{\ket{#1}#2\bra{#1}}
 \newcommand{\Project}[2]{\Ket{#1}#2\Bra{#1}}
 \newcommand{\expect}[1]{\braket{#1}}
 \newcommand{\Expect}[1]{\Braket{#1}}
}
{}
```

Here we load `siunitx` if the `units` option was passed.

```
\ifthenelse{\boolean{punits}}
{
 \RequirePackage{siunitx}
```

These are some useful non-SI units

```
 \DeclareSIUnit{\torr}{torr}
 \DeclareSIUnit{\mmhg}{mmHg}
 \DeclareSIUnit{\amu}{amu}
 \DeclareSIUnit{\yr}{yr}
 \DeclareSIUnit{\erg}{erg}
 \DeclareSIUnit{\gauss}{Ga}
 \DeclareSIUnit{\molar}{\textsc{M}} % this follows the style set up in the siunitx manual
 \DeclareSIUnit{\poise}{P}
 \DeclareSIUnit{\foot}{ft}
 \DeclareSIUnit{\mileperhour}{mph}
 \DeclareSIUnit{\pound}{lb}
 \DeclareSIUnit{\poundforce}{lbf}
}
{}
```

Now we come to assorted functions and keywords.

First the inverse trig functions. These are defined in the `mandi` package, but for backward compatibility I would like to give an informative error message if anyone tries to use them without loading `mandi`. The solution used here is to defer defining the functions until the end of the preamble, after all packages have been loaded.

```
\AtBeginDocument{%
```

At this point, if `mandi` has been loaded (or if some other package has defined these commands), the following `\providecommand`s will do nothing.

```
 \providecommand{\asin}{\PackageError{physymb}{inverse trig functions require the mandi package
 \providecommand{\acos}{\PackageError{physymb}{inverse trig functions require the mandi package
 \providecommand{\atan}{\PackageError{physymb}{inverse trig functions require the mandi package
 \providecommand{\asec}{\PackageError{physymb}{inverse trig functions require the mandi package
 \providecommand{\acsc}{\PackageError{physymb}{inverse trig functions require the mandi package
 \providecommand{\acot}{\PackageError{physymb}{inverse trig functions require the mandi package
```

Same for hyperbolic trig functions:

```
 \providecommand{\sech}{\PackageError{physymb}{hyperbolic trig functions require the mandi packa
 \providecommand{\csch}{\PackageError{physymb}{hyperbolic trig functions require the mandi packa
 \providecommand{\asinh}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
 \providecommand{\acosh}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
 \providecommand{\atanh}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
 \providecommand{\asech}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
 \providecommand{\acsch}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
 \providecommand{\acoth}{\PackageError{physymb}{hyperbolic trig functions require the mandi pack
}
```

Next are some linear algebra keywords.

```
\DeclareMathOperator{\diag}{diag}
\DeclareMathOperator{\realop}{Re}
\DeclareMathOperator{\imagop}{Im}
\newcommand{\real}[1]{\realop\{#1\}}
\newcommand{\imag}[1]{\imagop\{#1\}}
```

The absolute value and norm notations are implemented by `commath` so there is nothing to do for them here.

Evaluation at endpoints is implemented by either `commath` or `mandi`. As with the trig functions, we want to wait until the end of the preamble so that if `mandi` is loaded later, its definition of `\evalat` will remain. Otherwise, we implement it in terms of `commath`'s `\eval`.

```
\AtBeginDocument{%
 \providecommand{\evalat}[3]{\eval{#1}_{#2}^{#3}}%
}
```

If `mandi` is loaded, we can use its implementation of `\sgn`.

```
\AtBeginDocument{%
 \providecommand{\sgn}{\PackageError{physymb}{signum requires the mandi package}}
}
```

Same goes for `\orderof`.

```
\AtBeginDocument{
 \providecommand{\orderof}{\PackageError{physymb}{orderof requires the mandi package}}
}
```

Poisson brackets are just braces

```
\newcommand{\pbrac}[2]{\left\{#1,#2\right\}}
\newcommand{\pbracvars}[4]{\left\{#1,#2\right\}_{#3,#4}}
```

This handles the redefinition of `\vec`. If the `boldvectors` option was passed, a vector is denoted by bolding the argument. If `arrowvectors` was passed, the vector is denoted by putting an arrow over the argument. Some people use an undertilde, which will probably be added in the future.

```
\ifthenelse{\boolean{pboldvectors}}%
```

```
 {\renewcommand{\vec}[1]{\mathbf{#1}}}%
 {\renewcommand{\vec}[1]{\accentset{\rightharpoonup}{#1}}}
```

`\vecvar` is just a synonym for `\vec`

```
\newcommand{\vecvar}[1]{\vec{#1}}
```

`\tnsvar` always uses bold. Some people use undertildes, which will be added.

```
\newcommand{\tnsvar}[1]{\mathbf{#1}}
```

`\matvar` always uses bold.

```
\newcommand{\matvar}[1]{\mathbf{#1}}
```

`\identitym` is a bold 1

```
\newcommand{\identitym}{\mathbf{1}}
```

`\determinant` uses vertical bars.

```
\newcommand{\determinant}[1]{\envert{#1}}
```

`\trace` uses capital Tr.

```
\DeclareMathOperator{\trace}{Tr}
```

Now we get to unit vectors. This is just a wrapper for `\dirvect` from `mandi`.

```
\AtBeginDocument{
 \ifthenelse{\isundefined{\dirvect}}{%
  \newcommand{\unitvec}[1]{\PackageError{physymb}{unit vectors require the mandi package}}
 }{%
  \newcommand{\unitvec}[1]{\dirvect{#1}}
 }
}%
```

And these are a bunch of predefined unit vectors. Note that `\unitl` now uses a regular lowercase l, rather than the scripe ℓ as in previous versions. The new command `\unitell` will give you a unit ℓ.

```
\newcommand{\unitd}{\unitvec{d}}
\newcommand{\unite}{\unitvec{e}}
\newcommand{\uniti}{\unitvec{\imath}}
\newcommand{\unitj}{\unitvec{\jmath}}
\newcommand{\unitk}{\unitvec{k}}
\newcommand{\unitl}{\unitvec{l}}
\newcommand{\unitell}{\unitvec{\ell}}
\newcommand{\unitn}{\unitvec{n}}
\newcommand{\unitp}{\unitvec{p}}
\newcommand{\unitq}{\unitvec{q}}
\newcommand{\unitr}{\unitvec{r}}
\newcommand{\units}{\unitvec{s}}
\newcommand{\unitt}{\unitvec{t}}
\newcommand{\unitu}{\unitvec{u}}
\newcommand{\unitv}{\unitvec{v}}
\newcommand{\unitw}{\unitvec{w}}
\newcommand{\unitx}{\unitvec{x}}
\newcommand{\unity}{\unitvec{y}}
\newcommand{\unitz}{\unitvec{z}}
\newcommand{\unitphi}{\unitvec{\phi}}
```

```
\newcommand{\unitrho}{\unitvec{\rho}}
\newcommand{\unittheta}{\unitvec{\theta}}
\newcommand{\unitomega}{\unitvec{\omega}}
```

`\udc` is just an upright (roman) d, and similarly for higher-order differentials, implemented in terms of `\dif` from `commath`.

```
\newcommand{\udc}{\dif}
\newcommand{\uddc}{\dif^2}
\newcommand{\udddc}{\dif^3}
```

`\pdc` is just `\partial`, defined for similarity with `\udc`.

```
\newcommand{\pdc}{\partial}
\newcommand{\pddc}{\partial^2}
\newcommand{\pdddc}{\partial^3}
```

`\uds` is just like `\udc` but it includes a small space in front.

```
\newcommand{\uds}{\,\dif}
\newcommand{\udds}{\,\dif^2}
\newcommand{\uddds}{\,\dif^3}
```

`\pds` is also defined for similarity as just `\partial` with a space in front, although I'm not sure this one is really useful.

```
\newcommand{\pds}{\,\partial}
\newcommand{\pdds}{\,\partial^2}
\newcommand{\pddds}{\,\partial^3}
```

`\ud` typesets a derivative using `\udc`. Similarly for second and third derivatives.

```
\let\ud\od
\newcommand{\udd}[2]{\od[2]{#1}{#2}}
\newcommand{\uddd}[2]{\od[3]{#1}{#2}}
```

`\pd` (defined in `commath`) does the same for partial derivatives with `\pdc`.

```
\newcommand{\pdd}[2]{\pd[2]{#1}{#2}}
\newcommand{\pddd}[2]{\pd[3]{#1}{#2}}
```

All of `\div`, `\grad`, and `\curl` come from `mandi`.

```
\AtBeginDocument{
 \@ifpackageloaded{mandi}{%
  \let\grad\gradient
  \let\div\divergence
  \let\lapl\laplacian
 }{%
  \providecommand{\grad}{\PackageError{physymb}{gradient requires the mandi package}}
  % use renew instead of provide because \div is defined in plain latex
  \renewcommand{\div}{\PackageError{physymb}{divergence requires the mandi package}}
  \providecommand{\curl}{\PackageError{physymb}{curl requires the mandi package}}
  \providecommand{\lapl}{\PackageError{physymb}{laplacian requires the mandi package}}
 }
}
```

`\conj` just puts a superscript star

```
\newcommand{\conj}[1]{{#1 ^{*}}}
```

`\herm` is the same thing but for operators or matrices, so with a dagger

```
\newcommand{\herm}[1]{{#1 ^{\dagger}}}
```

`\transpose` does the same with a T

```
\newcommand{\transpose}[1]{{#1 ^{T}}}
```

These set notations are mostly done with `\mathbb`

```
\newcommand{\natset}{\mathbb{N}}
\newcommand{\intset}{\mathbb{Z}}
\newcommand{\cpxset}{\mathbb{C}}
\newcommand{\whlset}{\mathbb{Q}}
\newcommand{\realset}{\mathbb{R}}
\newcommand{\imagset}{\mathbb{I}}
```

Commutators and anticommutators are done in the obvious way

```
\newcommand{\commut}[2]{\left[ #1, #2 \right]}
\newcommand{\acommut}[2]{\left\{ #1, #2 \right\}}
```

The `\round` operator just typesets the word "round"

```
\DeclareMathOperator{\round}{round}
```

The exterior derivative is typeset in bold, in contrast to the differential d which is just a plain roman font

```
\DeclareMathOperator{\exd}{\mathbf{d}}
```

The Hodge dual uses a star, but not superscript like `\conj`.

```
\newcommand{\hodge}{\star}
```

These are short macros to typeset the symbols for the elementary (and common non-elementary) particles. They are defined in terms of commands from `heppennames`, but only if the `particle` option was passed.

```
\ifthenelse{\boolean{pparticle}}
{
\let\upq\Pqu
\let\dnq\Pqd
\let\srq\Pqs
\let\chq\Pqc
\let\btq\Pqb
\let\tpq\Pqt
\let\upaq\Paqu
\let\dnaq\Paqd
\let\sraq\Paqs
\let\chaq\Paqc
\let\btaq\Paqb
\let\tpaq\Paqt
\let\elp\Pem
\let\enu\Pgne
\let\ulp\Pgmm
\let\unu\Pgngm
\let\tlp\Pgtm
\let\tnu\Pgngt
\let\ealp\Pep
```

```
\let\eanu\Pagne
\let\ualp\Pgmp
\let\uanu\Pagngm
\let\talp\Pgtp
\let\tanu\Pagngt
\let\prbr\Pp
\let\nebr\Pn
\let\lmzbr\PgL
\let\sgpbr\PgSp
\let\sgzbr\PgSz
\let\sgmbr\PgSm
\newcommand\dlppbr{\HepParticle{\Delta}{}{++}}
\newcommand\dlpbr{\HepParticle{\Delta}{}{+}}
\newcommand\dlzbr{\HepParticle{\Delta}{}{0}}
\newcommand\dlmbr{\HepParticle{\Delta}{}{-}}
\let\xizbr\PgXz
\let\ximbr\PgXm
\let\ommbr\PgOm
\newcommand\sgspbr{\HepParticle{\Sigma}{}{*+}}
\newcommand\sgszbr{\HepParticle{\Sigma}{}{*0}}
\newcommand\sgsmbr{\HepParticle{\Sigma}{}{*-}}
\newcommand\xiszbr{\HepParticle{\Xi}{}{*0}}
\newcommand\xismbr{\HepParticle{\Xi}{}{*-}}
\let\prabr\Pap
\let\neabr\Pan
\newcommand\dlpabr{\HepAntiParticle{\Delta}{}{+}}
\newcommand\dlzabr{\HepAntiParticle{\Delta}{}{0}}
\newcommand\dlmabr{\HepAntiParticle{\Delta}{}{-}}
\newcommand\dlmmabr{\HepAntiParticle{\Delta}{}{--}}
\let\pipm\Pgpm
\let\pizm\Pgpz
\let\pimm\Pgpp
\let\kapm\PKp
\let\kazm\PKz
\let\kazam\PaKz
\let\kamm\PKm
\let\ropm\Pgrp
\let\rozm\Pgrz
\let\romm\Pgrm
\let\etam\Pgh
\let\etapm\Pghpr
\newcommand\kaspm{\HepParticle{K}{}{*+}}
\newcommand\kaszm{\HepParticle{K}{}{*0}}
\newcommand\kaszam{\HepAntiParticle{K}{}{*0}}
\newcommand\kasmm{\HepParticle{K}{}{*-}}
\let\omm\Pgo
\let\phim\Pgf
\let\phbsn\Pgg
\let\Wbsn\PW
\let\Wpbsn\PWp
```

```
\let\Wmbsn\PWm
\let\Zzbsn\PZ
\let\hbsn\Ph
\let\photon\Pgg
}
{}
```

The `feynman` option is implemented by just loading the package `feynmp`.

```
\ifthenelse{\boolean{pfeynman}}%
 {\RequirePackage{feynmp}}%
 {}
```