A common interface for quadrivectors and particles

Philippe Gras

Nov 5-6, 23

Université Paris-Saclay, CEA/IRFU - France

Introduction

- The Table interface from TABLES.JL is very nice and allowed interoperability of many packages (130+).
  - Key feature of his success is the simplicity of implementing it.
- Proposing to agree on a similar interface for Four-momentum and particle objects.
  - Particle object holds properties of a particle or particle candidate in an event: its nature (pdgid), momentum, vertex, charge, etc.
- Purpose: ease interoperability between the packages

The table interface example

To make a type compatible to the "Table" it is required to provide following methods:

`Tables.istable(::Type{MyTable}) = true`	Declare that the type implements the interface.
and one of	
`Tables.rowaccess(::Type{MyTable})`	Declare that it defines a `Table.rows(::MyTyple)` method.
`Tables.rows(x::MyTable)`	Return an `Tables.AbstractRow`-compatible iterator.
or	
`Tables.columnaccess(::TypeMyTable) = true`	Declare that the type defines `Tables.columns(::MyTable)` method.
`Tables.columns(x::MyTable)`	Return an `Tables.AbstractColumns`-compatible object.

In addition, 5 methods are optional.

See Tables.jl documentation for more details.

Note

The branching in the code,

```
if Tables.rowaccess(table)
  ...
elseif Tables.columaccess(table)
 ...
else
...
end
```

will be solved and removed by the compiler when optimizing the code $\Rightarrow$ no performance penalty.

Four-momentum: name usage survey

Proposal	LorentzVectorHEP	HepMC3	TLorentzVector	Rivet	QED.jl[1]	Fastjet
pt	pt	pt, perp	Pt, Perp	perp	getPt, getPerp, getTransverseMomentum	pt
	-	pt2, perp2	Perp2	perp2	getPt2, getPerp2, getTransverseMomentum2	pt2, perp2, kt2
eta	eta	eta, pseudoRapidity	PseudoRapidity	eta, pseudorapidity	-	pseudorapidity
rapidity	rapidity	rap	Eta, Rapidity	rap, rapidity	getRapidity	eta, rap, rapidity
phi	phi	phi	Phi	phi	getPhi	phi, phi_std, phi_02pi
energy	energy	e , t	E, Energy, T	pt, pT, t	getE, getEnergy, getT	e, E
mass	mass	m	M, Mag	mass	getMass, getInvariantMass, getMag, getMagnitude	m
px	px	px , x	Px, X	px, x	getPx, getX	px
py	py	py , y	Py, Y	py, y	getPy, getY	py
pz	pz	pz , z	Pz, Z	pz, z	getPz, getZ	pz

[1] Plan to drop the get prefix.

Four-momentum proposal i

To make a type compliant with to the "FourMomentum" interface, it is required to provide following methods:

`FourMomenta.isfourmomentum(::Type{MyP4}) = true`	Declare that the type implements the interface.
and one of	
FourMomenta.pxpypze(::MyType) = true px(::MyType), oy(::MyType), pz(::MyType), energy(::MyType)	Declare that px(), py(), pz, enery() are implemented and are the components with the fastest access.
or	
FourMomenta.pxpypzm(::MyFourMomentum) = true px(::MyType), py(::MyType), pz(::MyType), mass(::MyType)	
or	
FourMomenta.ptetaphim(::MyFourMomentum) = true pt(::MyType), eta(::MyType), phi(::MyType), mass(::MyType)	
or	
FourMomenta.ptetaphie(::MyFourMomentum) = true pt(::MyType), eta(::MyType), phi(::MyType), energy(::MyType)	

Default method implementations provided by the FourMomenta.jl package will compute the components in the alternative bases.

Four momentum proposal ii

- Base package kept minimal and utility functions e.g., ΔR, provided in another package.
- The type can optionally inherit from `AbstractFourMomentun` type to benefit from a default implementation of operators.
- Should the definition interval of `phi()` be specified? E.g., we could have a methods for $[0, 2\pi]$, one for $[-\pi, \pi]$, and one with no restriction.

An alternative approach

The Tim Holy's trait trick as I've understood it:

Use `Type{true}` instead of `true` as returned value of the flag functions, and then we can do a dispatch on it as follows.

```julia
f(p4) = f_(p4, FourMomenta.ptetaphie(::MyFourMomentum))
function f_(p4, ::Type{true}); ...; end
```

Any advantage or disadvantage with respect to former option?

Particle proposal

Particle interface need some thoughts.

- What do we want to store?
- Do we implement a `has<property>` method for each optional property?
- Do we use instead a *sentinel* value returned by the default method?

Example:

Mandatory:

- `isparticle(::MyType) = true`
- `momentum(::MyType)`, *the interface package can provide function px(::Particle),... shortcuts.*

Optional:

- `charge(::MyType)`
- `vertex(::MyType)`
- `spin(::MyType)`
- `isolation(::MyType)` or `isolated(::MyType)?`
- `b-tagged(::MyType)?`
- `hashistory(::MyType) = true`
- `mother(::MyType)`
- `daugthers(::MyType)`