Friedrich-Alexander-Universität
Erlangen-Nürnberg

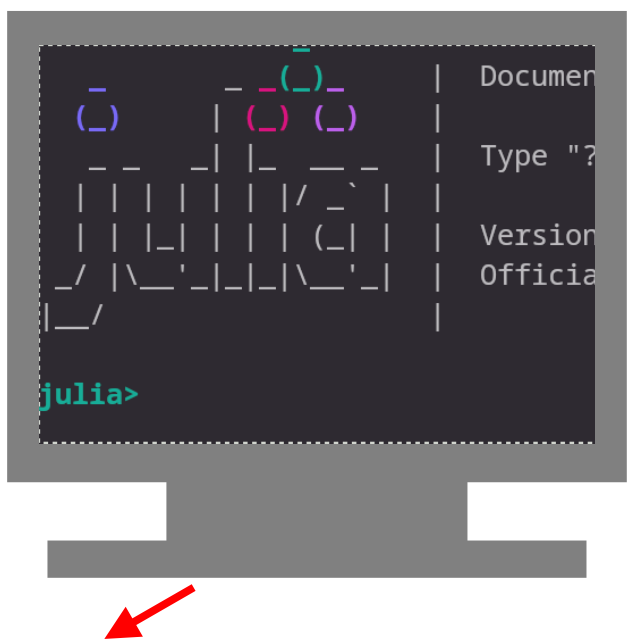
Corpuscles.jl

Johannes Schumann
JuliaHEP, 07 Oct 2023

The Problem

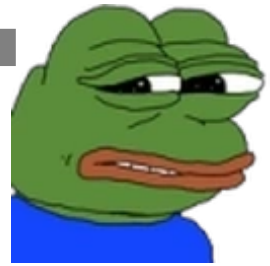
Imagine you work in HEP with Julia and you need some particle information …

… and your screens look something like this:



Lookup and
hardcode values

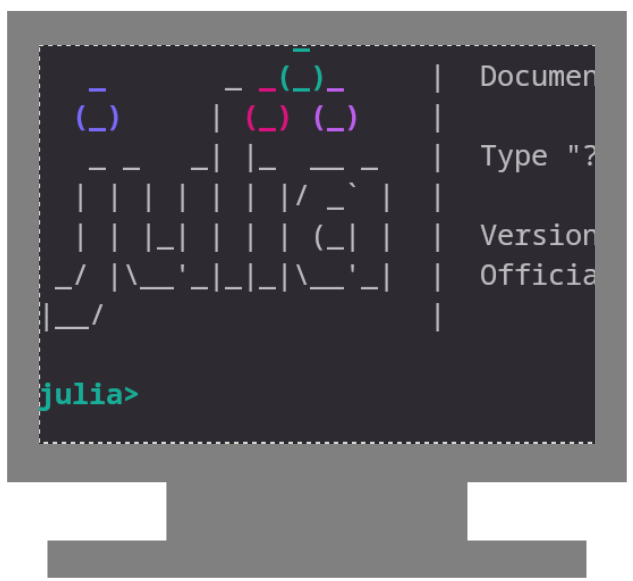
Particle

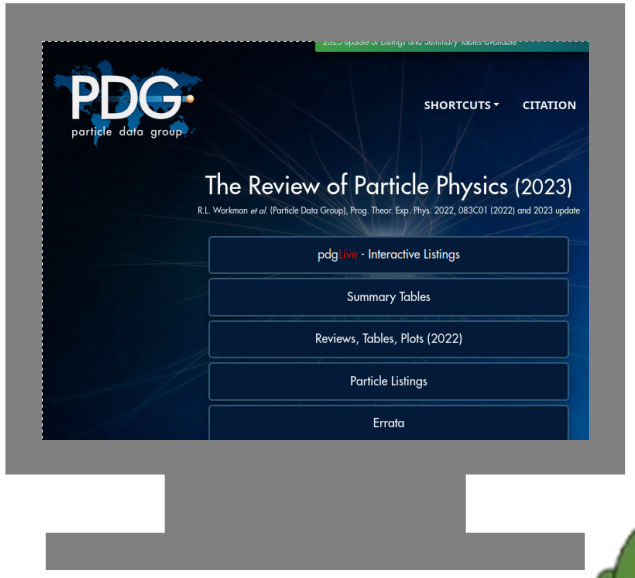


The Problem

Imagine you work in HEP with Julia and you need some particle information …

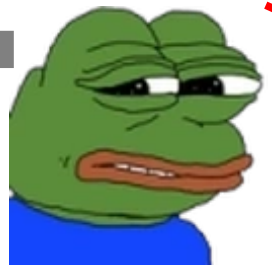
… and your screens look something like this:





Lookup and
hardcode values

Particle

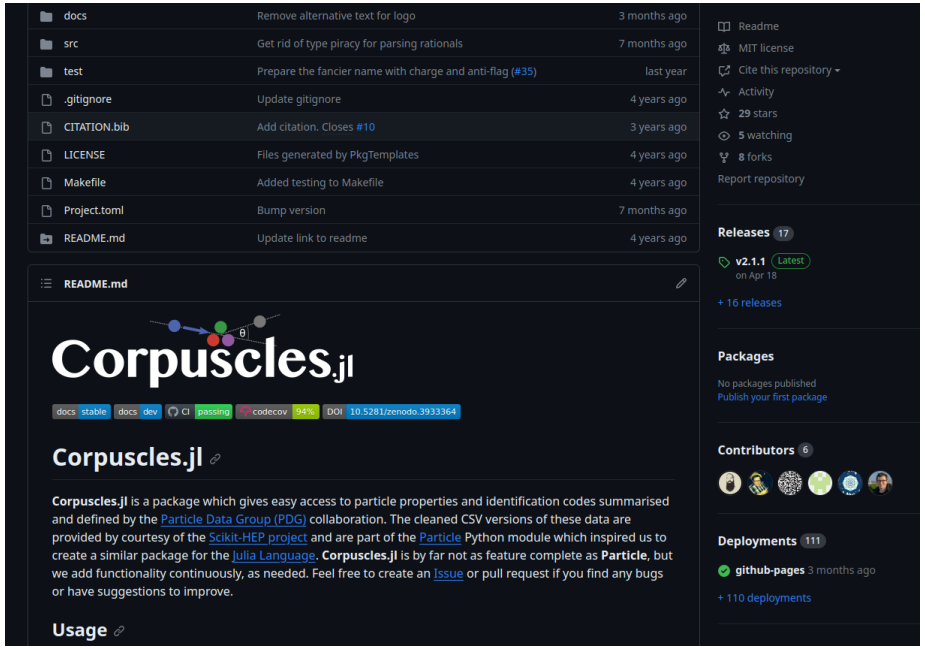


**Write an own
native julia package**

Corpuscles.jl in a nutshell

- Native Julia package
- Uses same CSV table like SciKit-HEP Particle
  → Official catalog slightly modified/fixed
- Mainly based on info encoded to PDGID
  → PDGdb.jl seems to provide a lot more additional info:
  e.g. decays and branching ratios …
- Values are given using **`Unitful.jl`**

GitHub



How does it work ...

VALUES

the start ...

```
julia> using Corpuscles

julia> p = Particle(12)
Particle(12) 'nu(e)'

julia> print(p)
Name:     nu(e)
PDG ID:   12
LaTeX:    $\nu_{e}$
Status:   Rare
Width = -1.0 MeV ± -1.0 MeV
Q (charge) = 0//1 e
Mass = -1.0 MeV ± -1.0 MeV
```

Mass & Width

```
struct MeasuredValue{D}
    value::Quantity{T1,D,U1} where {T1 <: Real, U1 <: Unitful.Units}
    lower_limit::Quantity{T2,D,U2} where {T2 <: Real, U2 <: Unitful.Units}
    upper_limit::Quantity{T3,D,U3} where {T3 <: Real, U3 <: Unitful.Units}
end
```

How does it work ...

VALUES

the start ...

```
julia> using Corpuscles

julia> p = Particle(12)
Particle(12) 'nu(e)'

julia> print(p)
Name:     nu(e)
PDG ID:   12
LaTeX:    $\nu_{e}$
Status:   Rare
Width = -1.0 MeV ± -1.0 MeV
Q (charge) = 0//1 e
Mass = -1.0 MeV ± -1.0 MeV
```

PDGID

```
struct PDGID <: ParticleID
    value::Int32
    Nj::Int8
    Nq3::Int8
    Nq2::Int8
    Nq1::Int8
    Nl::Int8
    Nr::Int8
    N::Int8
    N8::Int8
    N9::Int8
    N10::Int8

    function PDGID(value)
        d = digits(abs(value), pad=10)
        # splatting in `new()` is only supported in Julia 1.2+
        return new(value, d[1], d[2], d[3], d[4], d[5], d[6], d[7], d[8], d[9], d[10])
    end
end
```

How does it work ...

VALUES

the start ...

```
julia> using Corpuscles

julia> p = Particle(12)
Particle(12) 'nu(e)'

julia> print(p)
Name:     nu(e)
PDG ID:   12
LaTeX:    $\nu_{e}$
Status:   Rare
Width = -1.0 MeV ± -1.0 MeV
Q (charge) = 0//1 e
Mass = -1.0 MeV ± -1.0 MeV
```

PDGID

```
struct PDGID <: ParticleID
    value::Int32
    Nj::Int8
    Nq3::Int8
    Nq2::Int8
    Nq1::Int8
    Nl::Int8
    Nr::Int8
    N::Int8
    N8::Int8
    N9::Int8
    N10::Int8

    function PDGID(value)
        d = digits(abs(value), pad=10)
        # splatting in `new()` is only supported in Julia 1.2+
        return new(value, d[1], d[2], d[3], d[4], d[5], d[6], d[7], d[8], d[9], d[10])
    end
end
```

What the hell

Used by the helpers to check properties by the ID definition

How does it work ...

VALUES

the start ...

```
julia> using Corpuscles

julia> p = Particle(12)
Particle(12) 'nu(e)'

julia> print(p)
Name:     nu(e)
PDG ID:   12
LaTeX:    $\nu_{e}$
Status:   Rare
Width = -1.0 MeV ± -1.0 MeV
Q (charge) = 0//1 e
Mass = -1.0 MeV ± -1.0 MeV
```

Helpers

```
export Particle, PDGID, PythiaID, Geant3ID, particles

# helpers.jl
export isfundamental, isstandard
export isquark, islepton, ismeson, isbaryon, ishadron
export isRhadron, isSUSY, ispentaquark, isdyon, isnucleus, isdiquark
export istechnicolor, iscompositequarkorlepton
export isgaugebosonorhiggs, issmgaugebosonorhiggs
export isgeneratorspecific, isspecial, isQball, hasfundamentalanti
export hasdown, hasup, hascharm, hasstrange, hasbottom, hastop
export A, Z, charge, threecharge, J, S, L, jspin, sspin, lspin
```

Performance

What it looks on the first view:

```
julia> @benchmark Particle(ids).mass.value.val setup=(ids=sample(collect(keys(Corpuscles.PIDNames)), 1)[begin])
BenchmarkTools.Trial: 10000 samples with 222 evaluations.
 Range (min … max):  334.140 ns …   6.143 μs  ┊ GC (min … max): 0.00% … 91.41%
 Time  (median):     340.459 ns               ┊ GC (median):    0.00%
 Time  (mean ± σ):   357.961 ns ± 142.951 ns  ┊ GC (mean ± σ):  0.99% ±  2.41%
```

```
  334 ns        Histogram: log(frequency) by time        493 ns <

 Memory estimate: 160 bytes, allocs estimate: 2.
```

Performance

Compare to a dictionary:

```
julia> a = Dict{Int64, Int64}()
Dict{Int64, Int64}()

julia> for i in 1:1000
           a[i] = i
       end

julia> @benchmark a[ids] setup=(ids=sample(collect(1:1000),1)[begin])
BenchmarkTools.Trial: 10000 samples with 994 evaluations.
 Range (min … max):  32.486 ns …  1.676 μs  ┊ GC (min … max): 0.00% … 94.24%
 Time  (median):     34.502 ns              ┊ GC (median):    0.00%
 Time  (mean ± σ):   45.705 ns ± 33.365 ns  ┊ GC (mean ± σ):  1.29% ±  1.88%

  32.5 ns      Histogram: log(frequency) by time      80.1 ns <

 Memory estimate: 0 bytes, allocs estimate: 0.
```

Performance

The struggle mainly comes from splitting the number into its PDGID parts:

```
julia> @benchmark PDGID(ids) setup=(ids=sample(collect(keys(Corpuscles.PIDNames)), 1)[begin])
BenchmarkTools.Trial: 10000 samples with 930 evaluations.
 Range (min … max):  110.013 ns …   2.621 μs  ┊ GC (min … max): 0.00% … 90.05%
 Time  (median):     112.211 ns               ┊ GC (median):    0.00%
 Time  (mean ± σ):   122.811 ns ± 116.463 ns  ┊ GC (mean ± σ):  5.42% ±  5.47%

  110 ns        Histogram: log(frequency) by time        249 ns <

 Memory estimate: 176 bytes, allocs estimate: 2.
```

Hands-On

Presentation materials

- Corpuscles.pdf
- corpuscles_pluto.jl
- dummy_particles.csv

Pluto example with some functions in the Indico!

```
p = Particle(311) K0
```

```
p = Particle(311)
```

```
print(p)
```

```
Name:    K0
PDG ID:  311
LaTeX:   $K^{0}$
Status:  Common
Width = -1.0 MeV c^-2 ± -1.0 MeV c^-2
Q (charge) = 0//1 e
Composition = dS
Isospin = 1//2
Mass = 497.611 MeV c^-2 ± 0.013 MeV c^-2
P (space parity) = -1
```

```
0
```

```
L(p)
```

```
using CSV ✓
```

```
using DataFrames ✓
```

```
[PDGID(2112), PDGID(211), PDGID(-211), PDGID(211), PDGID(2212), PDGID(2112), PDGID(2212),
```

```
begin
    df = DataFrame(CSV.File("./dummy_particles.csv"));
    df[!,:PDGID] = PDGID.(df[!,:PDGID])
end
```

```
9999
```

```
nrow(df)
```

	entry	subentry	weight	PDGID	Px	Py	Pz	E
1	0	0	0.00041807	PDGID(2112)	-0.551422	0.0597869	0.819209	1.363
2	0	1	0.00041807	PDGID(211)	1.1174	-2.7202	19.7152	19.93
3	0	2	0.00041807	PDGID(-211)	0.133585	-0.99126	13.6911	13.72
4	0	3	0.00041807	PDGID(211)	0.0605794	0.108613	0.102389	0.212
5	0	4	0.00041807	PDGID(2212)	0.80852	-0.230798	0.158952	1.269
6	0	5	0.00041807	PDGID(2112)	0.0699436	0.255203	0.177112	0.990

Summary

- Native package for PDG definitions and MC numbering scheme
  → … nothing tooooo fancy, but very helpful
- **Planned Improvements / Outlook**:
  → Use sqlite database (by using `PDGdb.jl` ;)
  → … add more helpers
  → Expand it to whole nucleii
  … or make a smooth transition of the interface to `PeriodicTable.jl`
  → Completion of the docs: JOSS paper still todo

Friedrich-Alexander-Universität
Erlangen-Nürnberg

ERLANGEN CENTRE
FOR ASTROPARTICLE
PHYSICS

FAU

Thank You