DUNE System of Units and Config for Phlex Era

Brett Viren

August 21, 2026

Topics

- System of units (SoU).
- Proposal for DUNE to adopt CLHEP's SoU.
- Handling external packages with different units.
- Initial support packages.
  - `hep-units` system for C++, Python and Jsonnet.
  - `hep-config` solutions for the lesser languages.

What is a **system** of units

A set of **base units** (mm, ns, etc) spanning **dimension measures** (length, time, etc) **plus**:

- Required: **constants** representing all relevant units in terms of base units (`km`, `m2`)
- Bonus: **extensibility** to add new units and other constants (`c`, `G`, `hbar`).
- Bonus: **functions**, eg `parse("<value> <unit>") -> quantity`.
- Bonus: **types**, eg `quantity<length> len = 1*m;`.

Base units alone does not a system-of-units make.

The CLHEP system of units

Used by many HEP software packages (Geant4, ROOT, Gaudi, Wire-Cell Toolkit).

Provides:

- A particular set of **base units** relevant for HEP-scale quantities.
- A fairly exhaustive, if not fully complete set of **constants**.
- ~~Bonus: extensibility, functions and types.~~ Everything is a `double` idiom.

I propose DUNE adopts the CLHEP system-of-units in the Phlex era.

Two rules to use CLHEP-style system of units

Import: always multiply a **literal number** by a **unit constant** to set a **variable**.

```
double length = 1.0 * units::GeV;
```

Export: always divide a **variable** by a **unit constant** to get an **other-unit value**.

```
double baseline = 1300 * units::km;
std::cout << "The baseline is " << baseline/units::mile << " miles\n";
```

```
The baseline is 807.78255 miles
```

Corollary: never store the other-unit value inside the SoU domain code!

- Or, if pushed, at least mark it: `baseline_mile = baseline/units::mile;`

Units firewall around external packages (eg LArSoft)

Software that has different base units must be **firewalled** from code in the system-of-units.

- This is a rare case where a unit is multiplied to a **variable** instead of a **literal number**.

Eg, we must just know LArSoft length unit convention is cm.

```
double length = get_some_larsoft_length() * units::cm;
```

Even better, don't sprinkle `units::cm` but provide a wrapper function:

```
double length = from_larsoft_length(get_some_larsoft_length());
```

Where does units firewall code live?

Bracketing a Phlex algorithm's code body

- Inside, an algorithm we are free to do "whatever".
- At the boundary (input, output, config) we agree to "speak" in the SoU.

Data products from external products

It is reasonable to want to exchange LArSoft data products in Phlex.

- Tricky to balance "purity" (all in SoU) and efficiency (avoid needless copies).

Poses a small problem. A few approaches next.

Approaches to firewalling units in data products

Make new types and convert at units firewall boundaries (my preference)

This is an opportunity to fix issues with existing LArSoft data product types

- Eg, `recob::SpacePoint` + charge → `dune::SpacePoint`
- Costs an extra transformation when consumer is actually LArSoft code.

Note: DUNE data *lingua franca* already introduces the extra transformation.

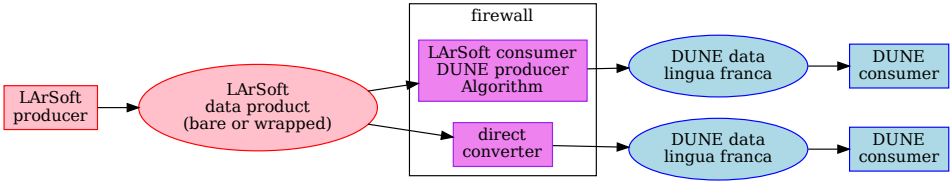
Mark external types as such in the Phex data-layer hierarchy (DLH)

Instead of passing bare LArSoft types in, wrap/mark them as outside SoU:

- Eg, `dune::external::Product<LarsoftType, LSUnits>`
- Binds external product with a type to help user convert values to DUNE's SoU.

Or, simply trust the algorithm developer to be disciplined when using bare LS types.

Hybrid / relaxed approach



Phlex graph firewall between external LArSoft and DUNE algorithm and data

We may relax the rules and push the firewall boundaries out from algorithm bodies.

But, we should require all configuration to be in the system-of-units!

hep-units - extensible, CLHEP SoU in multiple languages

https://github.com/brettviren/hep-units

- One time conversion of CLHEP SoU header to a simple text data file format.
- Provides a **data file parser** and a **multi-language code generator**.
  - Generates source files for **C++, Python, Jsonnet**.
  - Single-file products generated, committed to repo, can be “vendored”.
- Includes a `parse(<units string>) -> value` function for each language.
- Provides mechanism to extend to new units and physical constants.

Possible future hep-units upgrades

- Integration of Python codegen into Python `pint`.
- Provide library of firewall functions for LArSoft or other externals.

Phlex-induced problem: support for other than Jsonnet

Phlex support Jsonent, FHiCL, JSON and YAML.

- Jsonnet is a Real Programming Language(TM).
- FHiCL, JSON and YAML are barely data languages.
  - No support for variable reference nor arithmetic.

For not-Jsonnet, at best we can invent a **custom μDSL** and give units as strings:

- **good**: Jsonnet immediately gives us an actual number.

```
local units = import "units.jsonnet";
{ my_threshold: 100 * units.MeV }
```

- **bad**: JSON must give us a string for later interpretation.

```
{ "my_threshold": "100 * MeV" }
```

Your algorithm must handle this config babel!

Phlex gives your algorithm a dynamic `boost::json` object.

- **You** must check every attribute. Any missing/extra/wrong type?
- **You** must figure out what to do when an expected float is a string.

At least `hep-units` gives you `parse()`!

- But, can I provide more help?

Possible way out - side quest into boost-config

https://github.com/brettviren/boost-config

Let's you write documented config with auto JSON serialization and default field values:

```
#include "boost_config/boost_config.hpp"
using boost_config::Documented;

struct MyConfig {
    Documented<int> x{1, "Docstring for the attribute 'x'..."};
    float           y = 2.0f;   // undocumented field allowed
};

boost::json::value jv = ...; // Phlex passes it to your algorithm
auto cfg = boost::json::value_to<MyConfig>(jv);   // JSON -> struct
boost::json::value out = boost::json::value_from(cfg); // struct -> JSON
```

hep-config = hep-units + boost-config

- **boost-config** provides a “hook”, normally a no-op, that can process scalars in the JSON/C++ serialization.
- **hep-config** injects that hook by calling **hep-units** `parse()` on strings that should be floats.

Just tell me what I need to do with Phlex to have safe config!

```
#include "hep/units_boost_config.hpp"
#include "boost_config/Documented.hpp"
#include "boost_config/json.hpp"
#include <boost/json.hpp>

struct MyConfig {  // WRITE THIS
    Documented<int> x{1, "Docstring for the attribute 'x'..."};
    float           y = 2.0f;   // undocumented field allowed
};

class MyAlg {      // Your alg class as usual.
  MyAlg(MyConfig& cfg) { ... };  // No need for boost::json ctor
};

PHLEX_REGISTER_ALGORITHMS(m, jcfg) {
  auto cfg = boost::json::value_to<MYConfig>(jcfg); // add this
  auto exec = std::make_shared<MyAlg>(cfg);         // Phlex SOP
  m.transform("nodename", [exec](...) {...}, ...);
}
```

Going beyond “flat” units

CLHEP-style SoU flattens all units to `double`, erasing the units.

```
double len = 10*mm; double time = 10*ns;
double wtf = len + time; // CLHEP doesn't care
```

`mp-units` and `boost::units` carry units as C++ types, compile-time units errors.

```
quantity<force>     F(2.0 * newton); // Define a quantity of force.
quantity<length>    dx(2.0 * meter); // and a distance,
quantity<energy>    E(work(F,dx));  // and calculate the work done.
```

`mp-units` is the modern version of `boost::units`.

- If we adopt this `quantity` idiom, evaluate if `mp-units` is worth the extra dependency.

Summary

I have provided low-dependency packages supporting system-of-units and configuration:

- **hep-units** brings extensible, CLHEP SoU to C++, Jsonnet, and Python.
- **boost-config** serializes between C++ structs and Phlex' config type `boost::json`.
- **hep-config** combines the two to add support for `"<value> <unit>"` strings.

I propose DUNE uses a common **system-of-units** in the Phlex era.

- Adopt **hep-config** as the provider.

FIN

Backups

Why not use CLHEP-style SoU but with LArSoft base units?

Technically possible but…

- It's like measuring weight in stones or distance in miles.
- CLHEP is the "metric system" of (HEP) units.
- It would marginalize major packages which make up a majority set.

General Xerosere / Phlex integration status

What exists right now

- 21 C++ packages covering WCT, edep-sim, Arrow, HDF5, config, units.
  - Various Python util packages / displays and extended `phlex` C++ `main()`'s.
- PDHD and PDVD detectors with generic detector-parameterized Jsonnet (Phlex+WCT).
- Fullest chain: kine -> G4/edep-sim -> sim/WCT -> SP/WCT
- Standard jobs for WCT metrics: SPDIR (SP eff/bias/res).

Coming Real Soon Now

- edep-sim/Simphony GPU-accelerated optical photon tracking.
- "Morse metric" workflow to measure extra smearing required for depo-flux-splat.
- Wire-Cell imaging, clustering, pattern recognition data and algorithm integration.

Longer term

- Retry using FORM in 0.3.2+, waiting for phlex 0.4.0+ to try new "resources".