Wire-Cell Toolkit ↔ Phlex Framework

Initial integration work

Brett Viren

May 4, 2026

Topics

Wire-Cell Toolkit ↔ Phlex Framework
Initial integration

Brief Phlex history

- FNAL decided to replace *art* with an entirely new framework for DUNE.
- FNAL R&D funded Kyle Knoepfel to develop experimental **meld** framework.
- FNAL/DUNE is funding **Phlex** development based on meld.
- Two **Phlex** releases (0.2.0) to date, each with intentionally limited scope.

General Phlex issues for integration

Phlex (0.2.0) is **very bare bones**.

- No "real" data products, nodes/algorithms, job configuration.
  - This integration must invent a lot of things that `larwirecell` took for granted.
- Example node and data are all fairly trivial.
  - Trivial is good to have but more complex variations require invention.

But, we are in a green-field and first-mover advantage is in play!

General Phlex non-issues for integration

- Plex concept, design and code is generally high quality.
- Decent documentation given the newness.
- No real build issues (builds on Debian no problem).
- A solid Spack-based ecosystem is provided and works well.

WCT/Phlex impedance matches

Both implement data flow programming (DFP) paradigm

Can share the same TBB thread pool, (a big deal for MT running).
→ Their graphs can not be freely mixed.

Both support the Jsonnet configuration language

Both also support JSON. Phlex additionally supports YAML and FHiCL.
→ Care is needed to keep the two sets of Jsonnet distinct.

Very minimal requirements for defining Phlex edge data

Well, true for *art*, but `larwirecell` had to target existing LArSoft "data products".

- Minimal wrappers around WCT `IData` are still required.

WCT/Phlex impedance mismatches

Phlex is static typed, WCT is dynamic

- WCT Interface classes allow for small family of type abstractions.
- Phlex C++20 concepts allow an open-ended set of types, but statically typed.

→ Integration needs more code than in the case of art / `larwirecell`.

Phlex execution model not fully matching WCT

- Phlex nodes are more “function” than WCT’s “callable object”.
- No explicit construct-configure-execute-terminate-destruct phasing.

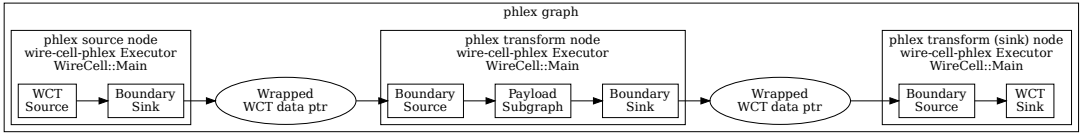
→ Mostly okay but may WCT service as wrapper of Phlex service will be tricky.

Phlex provides neither CLI config parameter settings nor config load path

- Open issue/discussion on Phlex GitHub to add TLA, `extVar()` and PATH like env var.

→ For now, I run WCT’s `wcsonnet` on Phlex Jsonnet to make Phlex JSON.

WCT / phlex integration overview



Approximately the same design as the WCT/art integration in `larwirecell`.

- Embed WCT-graph-as-phlex-node
- Data converters at boundaries.

Major differences from `larwirecell` case:

- Phlex hard-typing means more node/graph code in the integration.
- Phlex (lack of) data model means less data converter code.
- Phlex services (“resources”) are handled like data.

WCT data wrappers

To adhere to phlex C++20 “concepts” simple WCT data wrapper is sufficient:

```
struct Frame {
    WireCell::IFrame::pointer ptr;
};
```

Similar for `IDepoSet`, `ITensorSet`, etc and their constituents.

WCT/Phlex “boundary” data converters

Need “two-faced” converters, one face sees Phlex, the other sees WCT

- A thread-safe queue transfers data from one face to the other.

```
template <typename WctInode>
class BoundarySink : public WctInode { ... };
class BoundarySource : public WctInode { ... };
```

This same concept lives in `larwirecell` except here the converters become trivial and generated by C++ templates!

- Merely need template specializations for every supported `ISource` and `ISink` type.
- Will need “fat” converters if/when DUNE data model provides competing concepts.

Executor: a WCT graph as a Phlex node

Like with art, a WCT graph (not bare nodes) is the unit of framework execution.

- Unlike art, we must make **one Phlex type for every WCT sub-graph "shape"**.

Can lead to a **combinatoric type explosion**: $(node\ categories) \times (edge\ types)$

- Mitigation: limit support to most needed graph shapes.

```
// Phlex base class for all equiv to larwirecell WCLS_tool
class Executor { WireCell::Main m_wcmain; /*etc*/ };
// DepoSet->Frame
class DepoSetToFrame : public Executor {
    Frame operator()(DepoSet const& input);
}
// Frame->Frame
class FrameFilter : public Executor {
    Frame operator()(WireSchema const& ws, Frame const& input);
};
// etc for a just a few more
```

Status of `wire-cell-phlex`

For now, in: `https://github.com/brettviren/wire-cell-phlex`

- CMake-based build that depends on WCT and Phex.
- Provides `Executor` harness and `Boundary` converters.
- Includes Phlex+WCT configuration derived from `dunereco` for PDHD and PDVD.
- Wrapped WCT data sent through Phlex graph as first-class objects.
- Good unit test coverage using CMake's `ctest`.
- Some documentation: README and other markdown files.
- Uses SPDIR Snakefile workflow as integration test driver
  - $depos \to \{splat, \{sim \to sigproc\}\} \to metrics \to plots$
  - File I/O provided by WCT.
  - Results look reasonable.

Some known TODOs

- Add `wire-cell-phlex` packaging to `wire-cell-spack`.
- Get WC core folks to try to break things.
- Extend support, config and testing for downstream reco (imaging, clustering, etc).
- Develop plan for how to introduce `wire-cell-phlex` to broader DUNE.

$\mathcal{FIN}$

Backups

Integration workflow - heavily leverage LLM (Claude Sonnet + Opus)

1. Provide work area with git clones of `wire-cell-{toolkit,python,data,spack}`, `phlex{,-design,-examples}`, `larwirecell`, `dunereco`.
2. Hand write detailed `initial-prompt.md` to illicit a "plan" from Opus.
   - Describe a phased development process
     1. Examine and summarize existing code (`docs/*-summary.md`)
     2. Present design options and iterate to make decisions (`docs/design-options.md`)
     3. Break down implementation into steps after which a human can test and potentially change the plan.
   - Lengthy prose describing WCT and Phlex differences and similarities
3. Iterated all in one `claude` session working partly from written plan and my own mental plan.
   - Started a `docs/future.md` by hand with problems found as we went.
   - LLM found it and solved problems listed and updated the doc with solutions without being asked.
   - Later dev stages driven by fleshing specific jobs, asked for a summary after each (`docs/*-job.jsonnet` and special topics like, `executors.md`
4. Follow-on work on the `spdir-phlex` workflow to drive further refinement, validation and configuration.
   - Multi-source extraction/synthesis to produce config from `dunereco` that normalizes detector names and factors detector-specific objects of common schema from detector-generic job configuration.
   - Along the way, had to upgrade `wire-cell-python` to add PDVD support and some of this same kind of detector independent/specific factoring.

Calendar time from green field to current state: 3 days.

Integration workflow self-critique

- Staying in a single `claude` session the entire time was token-wasteful.
- The interactive human-influenced plan-following was effective but not well reproducible.
  - Better to have taken a task-graph orchestration (a'la beads) approach.
- Major time sinks came from unexpected sources.
  - PDVD ADCs were not found, likely due to some (still unknown) issue with detector coordinates.
  - I spent hours trying to find/fix and gave it to LLM which took 30 minutes.
- Some time wasted due to human-error driving my own tools (eg qtpc).
- Working on remote wcgpu and local workstation gave friction.
  - Better I could work fully local, except my local computer is old and slow.
- Time wasted for claude waiting for me and somewhat vice versa (I could multi-task a little).
  - Running claude in a container would give me confidence to dangerously skip permissions.