Wire-Cell Toolkit Data Interfaces

Brett Viren

April 10, 2024

Topics

- Understanding Wire-Cell Toolkit's (WCT) current internal and external I/O.
- Highlighting some pressing needs for I/O between WCT and *art* / LArSoft.
- Experimental and future I/O directions.

Categories of data interfaces in/with Wire-Cell Toolkit

Object

Intimate exchange of C++ objects in memory

- Inside WCT data flow graph
- Between WCT and `art::Event` or other external "services".

File

Serialization between objects and byte streams.

- General and special purpose files.
- Network sockets.

Categories of data inside Wire-Cell Toolkit

Configuration

Data used to define WCT's behavior.

- The `WireCell::Configuration` aka `JsonCPP::Value` object.
- Jsonnet/JSON file formats.
- Augmented by info from CLI and/or FHiCL.

Operational

The "working data", *eg* objects from detector/simulation/reconstruction.

- The `IData` object in WCT.
- Various supported file formats.
- Send to / get from external source/sink such as `art::Event`.

Focus on **operational data** now.

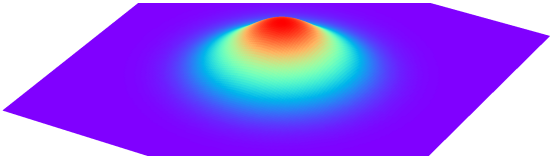
`IData` - interface to all operational data

All WCT data is accessed via (abstract) subclass of abstract interface `IData`.

- An `IData` is passed between WCT data flow graph nodes as `const` shared pointer.
- Any I/O with external software or files is performed by a DFP node.
  - *eg* via a “sink” or a “source” node.

`IData` objects are the “nouns”, nodes the “verbs” in WCT’s data-flow programming “grammar”.

IDepo



A group of ionization electrons centered at point and with Gaussian extent.

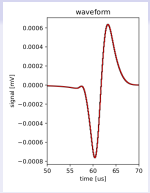
- Mostly consists of a 7-tuple: t, q, x, y, z, $\sigma_{\parallel}$, $\sigma_{\perp}$
  - Typically first born with $\sigma_{\parallel} = \sigma_{\perp} = 0$, diffusion leads to non-zero extent.
- Additional info to express IDs and association between pre- and post-drift depos.
- `IDepoSet` is a time-ordered collection of `IDepo`.

IFrame and ITrace

ITrace

Represent one **waveform fragment**

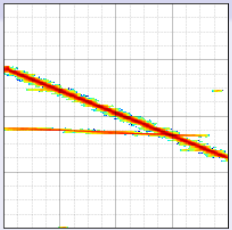
- Associated with one channel over a time duration.
- Channel ID, starting time bin and array of 32 bit FP samples.
- Support dense or sparse coverage of time.



IFrame

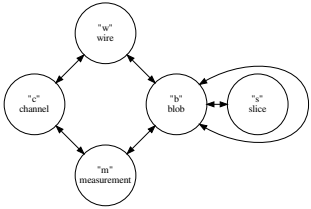
Represent a **collection of traces** across channels.

- Reference time, sampling period (“tick”), and collection of traces.
- “Tags” can ID frame, subsets of traces or regions in channel-tick space.
- Tagged traces can also have per-trace scalar “summary” values.
- Variable scope: single channel, single plane, single or multiple “APA”.



ICluster

Records the intermediate and final results of 3D imaging.

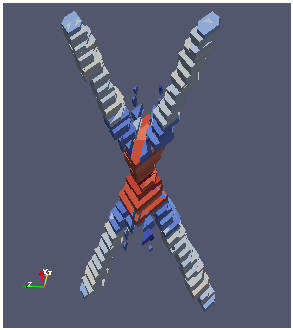


`ICluster` provides a graph connecting five `IData` vertex types:

- `IChannel` identifies electronics channel.
- `IWire` gives wire endpoints and ID.
- `ISlice` collects the signal fragments spanning a period of drift time.
- `IMeasure` signal sum across a contiguous set of channels in a plane and time slice.
- `IBlob` describes a 2D region limited by the measures of 3 views.

These objects also have internal inter-references.

- It is a somewhat complex beast!

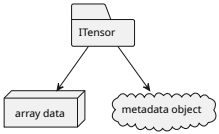


ITensor - a general purpose data structure

An `ITensor` consists of two elements:

- A N-dim **array** represented as `boost::multi_array`
  - shape, type, element size, memory order
- A **metadata** object represented as WCT `Configuration` object
  - aka `JsonCPP::Value`.

A set of `ITensor` can represent essentially any data structure.



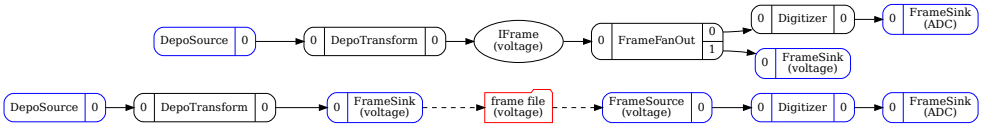
Schema and files.

- **Cluster graph schema** used to store `ICluster` as JSON, defined in a formal JSON Schema document. One giant JSON object.
- **Cluster array schema** used to store `ICluster` as Numpy arrays. Defined in a (human) spec document. Stores edge connections and the attributes of each type of node as individual arrays. Arrays are identified by name convention. Intentionally matches `torch_geometric.HeteroData` commonly used for GNN AI/ML.
- **Tensor data model** (TDM) maps `IData` to `ITensor` to file. Generic but highly "normalized" (in DB sense) so "some assembly required" to translate between TDM and working objects. Currently support **streams** of JSON+Numpy files to/from tar/zip/npz archives. HDF5 support planned.

Others: Bee JSON, Paraview VTK, Magnify ROOT, Celltree ROOT.

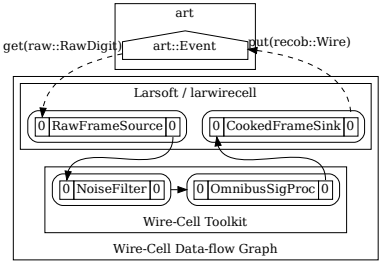
WCT SIO package: simple/streamed (file) I/O

Provides file I/O **source** and **sink** nodes for WCT formats/schema.



- Serialize DFP edge data to “cap off” or “tap into” the graph.
- A graph may be “sliced in half” so its intermediate data saved and later replayed.

larwirecell: *art* / LArSoft I/O



`larwirecell` provides some "two faced" components

- Appears to a WCT DFP graph as a sink or a source.
- Component may "visit" `art::Event` before and after graph execution

Correspondence between prominent Wire-Cell types and LArSoft types

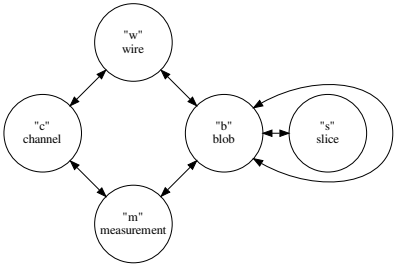
Wire-Cell	closest LArSoft	match
`IDepo`	`SimEnergyDeposit`	good
`IFrame`	`raw::RawDigit`'s	good*
`IFrame`	`recob::Wire`'s	good*
`ICluster`	`recob::SpacePoint`	poor
`ITensor`		none?

*Caveat: user must define how to map information held by `IFrame` "tags" to some kind of structure in `art::Event`.

larwirecell: support for `ICluster` needed!

Challenges in “exporting” the 3D imaging to *art* / LArSoft.

- `ICluster` is a rich, complex object with almost no overlap with current LArSoft data types.
- Additional internal (non-graph) inter-referencing makes it challenging to simply “port” `ICluster` types.
- Some information (eg, wire/channel) is in LS in other forms (eg geometry service). It is not good to duplicate this data in `art::Event`.



We want a data structure faithful to the info in `ICluster` but impedance-matched to the LArSoft data ecosystem.

Proposal: a dynamic LArSoft data product: `recob::Dataset`

Develop **general** and **dynamic** data product

- Represent the complexity of WCT 3D imaging and future pattern recognition results.
  - Allow simplifying and clarifying over complex `ICluster`
- Take inspiration from HDF5, Numpy arrays and `torch_geometric.HeteroData` data models (and make their I/O easy).

Basic concept of `recob::Dataset`

- Essentially, an "in memory" HDF5: groups of datasets (arrays) and metadata.
- Very similar to WCT `ITensor` so easy to apply for that driving goal.
- General data structure so can be useful for others.

Some issues

- Dynamic structure requires schema (implicit or explicit) and validation/interpretation.

AI/ML I/O via `TorchService`

`TorchService` is an `ITensorForward` implementation for PyTorch/libtorch

- Acts as a "service" to execute `forward()` on a Torch "module" on CPU or GPU.
  - Torch "module" provided as TorchScript which is $\approx$ Python.
- Thread safe and honors a WCT semaphore to limit outstanding tasks.
  - Avoid GPU/CPU overload given tasks initiated from multi-thread of multi-process WCT.

DNNROI DFP node runs AI/ML inference for signal-ROI

- Converts `IFrame` to `torch::Tensor` and feeds to `TorchService`
- Reverse transform on AI/ML output for resulting `IFrame`

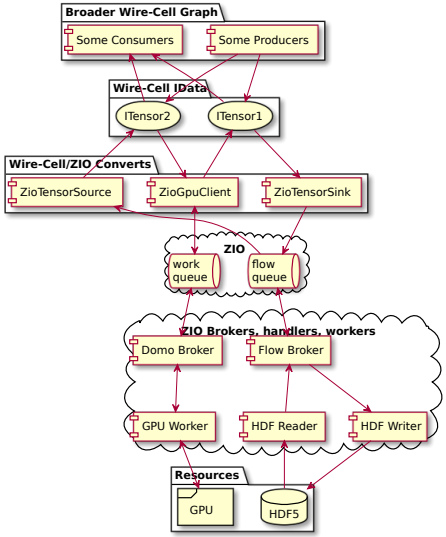
WCT also has an `IDFT` implementation to run Torch's FFT on CPU/GPU.

Distributed I/O - WCT ZIO

Enables some likely useful scaling:

- **GPU-as-a-service** sharing GPU with many, and varied clients.
- **Wire-Cell-as-a-service**, *eg* for triggered prompt supernova-ν burst pointing reconstruction.
- Exchange data "live" between WCT's and/or where hard-linked application not feasible (eg DAQ).
- Distributed, streamed, parallel data store, *eg* parallel read/write to HDF5 (w/out MPI).

Key ZeroMQ protocols and technologies: **majordomo** for tasks, **credit-based flow control** for streaming and **real-time exchange** (Zyre/ZRE) for discovery.



FIN

SIO: data types supported

`WireCellSio` plugin serializes the major types:

- `IDepo` as simple Numpy files (not yet documented in the tensor data model).
- `IFrame` as simple Numpy files and as tensor data model.
- `ICluster` as cluster graph (JSON) or cluster array (Numpy compatible with `torch_geometric.HeteroData`) or tensor data model.
- `ITensor` as tensor data model (JSON+Numpy in tar/zip/npz), future in HDF5.