# Frame files

Brett Viren

July 23, 2024

## Contents

**1 Intro** **1**

**2 Frame files** **2**

**3 Frame tensor files** **2**

3.1 Set-level . . . . . 3
3.2 Tensors . . . . . 4
3.3 Trace . . . . . 4
3.4 Index . . . . . 4
3.5 Summary . . . . . 5

## 1 Intro

A WCT "frame" provides a simple representation of data from different types of "readout" of a LArTPC detector with some additional concepts bolted on. This document describes briefly the explicit `IFrame` representation and then describes a number of other useful, related representations of frame data.

An `IFrame` holds an `ident` which is an integer often used to represent an "event number". The contents of an `IFrame` are located by its reference `time` and its sample period duration or `tick`. `IFrame` also holds one or more "frame tags" which are simple strings providing application-defined classifiers. The rest of the frame consists of three collections. The most important of which is `traces` which is a vector of `ITrace`. The second is a "trace tag map" with string keys interpreted as "trace tags" and values providing an array of indices into the `traces` collection. This provides application-defined classification of a subset of traces. Finally, a "channel mask map" with string keys interpreted as "mask tags" and values providing a mapping from a channel ident number to a list of tick ranges. A channel mask map is usually used to categorize regions in the channel vs tick domain as, for example, "bad".

The `ITrace` holds an offset from the frame reference `time` measured in number of `ticks`, a channel ident and a floating point array, typically identified as holding an ADC or a signal waveform (or fragment thereof). The `traces` array can be thought of as a 2D sparse or dense array spanning the channels and ticks of the readout.

## 2 Frame files

WCT `sio` subpackage provides support for reading and writing *frame files*. The format is actually a stream or archive of Numpy `.npy` files. The container format may be Zip (`.zip` or `.npz` file extension) or Tar with optional compression (`.tar`, `.tar.gz`, `.tar.xz`, `.tar.bz2`).

This file represents a decomposition of `IFrame` data into a number of arrays in `.npy` files and with a file naming scheme used to encode array type, frame `ident` and tag information. An expected stream of file names might look like:

```
frame_<tagA>_<ident1>.npy
channels_<tagA>_<ident1>.npy
tickinfo_<tagA>_<ident1>.npy
summary_<tagA>_<ident1>.npy
frame_<tagB>_<ident1>.npy
channels_<tagB>_<ident1>.npy
tickinfo_<tagB>_<ident1>.npy
frame__<ident1>.npy
channels__<ident1>.npy
tickinfo__<ident1>.npy
frame_*_<ident1>.npy
channels_*_<ident1>.npy
tickinfo_*_<ident1>.npy
chanmask_<cmtagC>_<ident1>.npy
chanmask_<cmtagD>_<ident1>.npy
...
```

We will call "framelet" the trio of arrays of type `frame`, `channels` and `tickinfo` and an optional fourth `summary` array that carry the same `<tag>` and frame `<ident>`.

To be valid, the size of the 1D `channels` array, and the summary array if present, must be the same as the number of (channel) rows in the `frame` array. The first two entries (`time` and `tick`) of all `tickinfo` must be be identical but their third entry (`tbin0`) may differ for each framelet. See `tickinfo.org`.

The rows of the `frame` array and the corresponding elements of the `channel` array are added to the `IFrame` traces collection in the order they are encountered in the file. Otherwise, order does not matter except that all arrays for a given frame `<ident>` must be contiguous.

The `<tag>` portion of the framelet provides a **trace tag** (and **not** a frame tag). When the `<tag>` is unspecified (ie, there are a pair of subsequent underscores, eg `frame__0~`) or when the `<tag>` takes a special value of `*` (eg `frame_*_0`) then the arrays are interpreted to represent an portion of the frame consisting of untagged traces. In such a case, any associated `summary_` is ignored.

The `chanmask` arrays are optional and each provides one entry in the `IFrame` channel mask map. The key is take as the `<cmtag>` part of the file name.

Despite the limitations of being both info-lossy and potentially data-inflating, this format is convenient for producing per-tag dense arrays for quick debugging using Python/Numpy.

## 3 Frame tensor files

A second format called *frame tensor files* improves on the above by mapping the frame data model to the one supported by `ITensorSet/ITensor`. This data model is also adopted by `Dataset/Array`

classes in WCT point-cloud support and it closely mimics the model implemented by HDF5. It further provides flexibility to the user by mapping the frame to the data model in one of three modes:

**sparse** traces are saved to individual array files, potentially of heterogeneous sizes.

**unified** all traces are saved in a single, zero-padded array file.

**tagged** multiple zero-padded arrays associated with a tag are saved.

The **sparse** mode is info-lossless and allows files to be small as possible, even without (or especially without) file compression. However, it stores each trace as an individual array and so processing is required to assemble the traces into a dense, padded 2D array.

The **unified** mode provides a single, dense, padded 2D array spanning the channels and ticks of all traces and with trace samples added onto the array. The padding and summing is info-lossless for details such as the original time span of a trace and the distinction of overlapping traces. On the channel axis the array may represent a conglomeration of multiple tagged sets of traces. If file compression is employed, the size is competitive with compressed files from sparse mode.

Finally, the **tagged** mode provides a similar representation to that of the original *frame file* format but it avoids the frame/trace tag ambiguity. Like *frame files*, traces in more than one tagged set are duplicated and like "unified" mode, padding and summing occur. Unlike both "sparse" and "unified" mode, any traces which are not in a tagged set are not persisted.

The remainder of this section describes how a frame is decomposed into the tensor data model. When applied to frames, this model provides for info-lossless persistence and potentially with no redundancy (strictly, only in "sparse" mode).

The WCT *frame tensor files* share some similarity with WCT *frame files*. They are also provided as streams or archive files in format of Zip or Tar with optional compression. Also like *frame files* the *frame tensor files* contain `.npy` files to hold array information. However, they also contain `.json` files to hold structured metadata. Furthermore the names of the individual file members of the *frame tensor files* archive/stream carry more general semantics.

Another difference is that serialization between `IFrame` and frame tensor file requires additional component in the WCT data flow graph compared frame file. A writing path in the graph might look like:

```
(IFrame) -> [FrameTensor] -> (ITensorSet) -> [TensorFileSink] -> file
```

A reading is simlar but reversed

```
file -> [TensorFileSource] -> (ITensorSet) -> [TensorFrame] -> (IFrame)
```

### 3.1 Set-level

The `ITensorSet` class and its metadata accepts frame information which is not dependent on trace-level information. To start with, the `IFrame::ident()` is mapped directly to `ITensorSet::ident()`.

Then the following shows the correspondence between `ITensorSet`-level metadata attribute names and the `IFrame` methods providing the metadata value:

`time` `IFrame::time()` float

`tick` `IFrame::tick()` float

`masks IFrame::masks()` structure

`tags IFrame::frame_tags()` array of string

When the set-level metadata is represented as a JSON file its name is assumed to take the form `frame_<ident>.json`. When `IFrame` data in file representation are provided as a stream, this file is expected to be prior to any other files representing the frame. The remaining files are expected to hold tensors and must be contiguous in the stream but otherwise their order is not defined. These tensors are described in the remaining sections.

### 3.2 Tensors

An `ITensor` represents some aspect of an `IFrame` not already represented in the set-level metadata. Each tensor provides at least these two metadata attributes:

`type` a label in the set `{trace, index, summary}` identifying the aspect of the frame it represents.

`name` an instance identifier that is unique in the context of all `ITensor` in the set of the same `type`.

The values for both attributes must be suitable for use as components of a file name. File names holding tensor level array or metadata information are assumed to take the forms, respectively `frame_<ident>_<type>_<name>.{json,npy}`.

The remaining sections describe each accepted type of tensor.

### 3.3 Trace

A trace tensor provides waveform samples from a number of channels. Its array spans a single or an ordered collection of channels. A single-channel trace array is 1D of shape `(nticks)` while a multi-channel trace array is 2D of shape `(nchans,nticks)`. Samples may be zero-padded and may be of type `float` or `short`. The ident numbers of the channels is provided by the `chid` metadata which is scalar for a single channel trace tensor and 1D of shape `(nchans)` for a multi-channel trace tensor.

- `tbin=N` the number of ticks prior to the first tensor column
- `chid=<int-or-array-of-int>` the channel ident number(s)
- `tag="tag"` an optional trace tag defining an implicit index tensor

If `tag` is given it implies the existence of a collection of tagged trace indices span the traces from this trace tensor. See below for how to explicitly indicate tagged traces.

`IFrame` represents traces as a flat, ordered collection of traces. When more than one trace tensor is encountered, its traces are appended to this collection. This allows sparse or dense or a hybrid mix of trace information. It also allows a collection of tagged traces to have their associated waveforms represented together.

### 3.4 Index

A subset of traces held by the frame is identified by a string ("trace tag") and its associated collection of indices into the full and final collection of traces.

`tag="tag"` a unique string ("trace tag") identifying this subset

### 3.5 Summary

A trace summary tensor provides values associated to indexed (tagged) traces. The tensor array elements are assumed to map one-to-one with indices provided by an index tensor with the matching tag. The additional metadata:

`tag="tag"` the associated index trace tag.

Note, it is undefined behavior if no matching index tensor exists.