2015

Specifications for Transformations between NSCLDAQ Data Formats

TOMPKINS, JEROMY

LAST MODIFIED: 2/11/2015

1 CONTENTS

2 INTRODUCTION TO THE DOCUMENT:

The goal of this document is to detail the consequences of transforming data between data formats. At the time of writing, there are only three data formats that exist whose versions will be referred to as 8.0, 10.0, and 11.0. These are all by nature incompatible with each other and transformations between them are nontrivial. To simplify the description of consequences, only transformations between adjacent data format versions will be discussed explicitly. The consequences of transforming between non-adjacent data format versions can be understood by considering the effect of applying multiple adjacent transforms. Though this may not be exactly how all transformations are implemented in practice, the consequences will be correct.

In the following sections the reader will be presented with all of the information required to understand conversions between adjacent formats. First, an overview of the guiding conversion principles will be presented because they inform the subsequent sections. Following that is a description of all three existing data formats with sufficient detail for the reader to grasp exactly what the inputs and outputs of the transformations will be. Once the reader fully understands the nuances of the various data formats, the actual transformation details will be described for conversions between adjacent data formats. Finally, having been provided a meaningful description of the transformations between data formats, some concluding remarks will be made.

3 GENERAL PRINCIPLES FOR CONVERTING BETWEEN FORMATS

Because the various data formats are incompatible with one another, some criteria must be used when defining the properties of transformations between data formats. It is assumed that the main reason conversion capabilities are desired is to allow analysis codes written for data in a specific data format to be used on data in other formats. The following five principles form a conversion strategy that favors this analysis code reuse for data read from front-end electronics.

Principle 1: Information that frames and describes blocks of data may not be preserved.

Any data format consists of both structural data and content data. The structural data defines the size of each block of related content as well as how to interpret its substructure. Structural data is tightly coupled to a specific data format by nature and has little to no meaning outside of its format version.

Principle 2: Content supported by the input format but not by the output format will be discarded.

Most types of data that would be considered fundamental to a nuclear physics experiment are supported by all data formats, but there may be data elements that are embodied in the input format that have no corresponding element in the output format. For example, the 11.0 format contains an optional data source identifier and an optional timestamp; the 10.0 format does not. If the source id and timestamp are present in the 11.0 data item, these will be discarded when transforming it to the 10.0 format. <u>This principle precludes inverse transformations</u>.

Principle 3: Content supported by the output format but not by the input format will be given sensible defaults

This serves as the opposite scenario to the previous principle. As an example, when transforming a data item in the 10.0 format to the 11.0 format, the optional body header that contains the timestamp and source id will be omitted.

Principle 4: Data elements may transform in ways that safely conserve information content

For some elements of the transmitted data format, the information content is more important than the memory layout. A relevant example of this is the run number. In some formats, the run number is stored as a 16-bit integer and in others it is stored as a 32-bit integer. It is assumed that the user cares little about how many bytes are allocated to store this number so long as the number remains the same.

Principle 5: Data blocks whose internal structure is determined by the specifics of each experiment will transform invariantly

This is a corollary of the previous principle but is worth stating explicitly. The structure of certain blocks of data in each format is intentionally left undefined. The actual structure of these sections will be determined by external logic, such as experiment-specific code or front-end electronics. Without knowing the logic that defines the structure of these sections, it is impossible to guarantee the information content is preserved without treating the blocks of data as invariants.

4 OVERVIEW OF DATA FORMATS

The content of this section aims to initiate the reader to the three NSCLDAQ data formats. Without sufficient understanding of these, it is impossible to grasp the issues that must be handled when transforming content embodied in one format to another. These descriptions only define the format as the data would be streamed between processes or stored on disk. To help the reader associate the principles laid out in the previous section with each data element, the elements have been color coded. The coding is as follows:

GREEN – structural data (meaning only relevant to current format version)

PURPLE – transmitted content data, informational content must be preserved if the output format supports the data element.

RED – transmitted content, invariant element. (Transformations must leave memory layout completely unchanged.)

4.1 VERSION 8.0

4.1.1 Overview

The 8.0 data format version is centered on a protocol that streams fixed size, buffers of homogenous content between processes. The default buffer size is 8192 bytes, so it is commonly referred to as the “8k buffer” format. The first 28-bytes of the buffer (“header”) contain a standard structure of information. The subsequent section of memory (“body”) follows immediately and contains data whose character is labeled by the type identifier in the header. The body must consist of a series of complete data structures whose type identifier is the same. For example, a buffer of physics data (type=DATABF) would have a single header followed by a series of physics data structures, each containing data from a single triggered readout. As many of these structures will be present as will fit into the 8k buffer. At the same time, a buffer containing the information for a begin run transition (type=BEGRUNBF) may only have a single control data structure and nothing more. The 8.0 data format is identified by the 8k buffer, but that is mainly just a protocol level detail for sending the data between processes. The actual amount of useful data in any given buffer was typically less than 8k and the format of the data within a buffer was dependent on the type identifier of the buffer.

The most common size for a piece of information in the 8.0 format is 16 bits. Any byte ordering information is provided explicitly in the buffer header by means of two byte order signatures.

4.1.2 Top-level Memory Layout (the “8k buffer”)

	Description	Size (bytes)	Offset (bytes)
Header	Content size (16-bit units)	2	0
	Type	2	2
	Checksum	2	4
	Run number	2	6
	Sequence number	4	8
	Number of events	2	12
	# LAM masks	2	14
	Processor number	2	16
	# bit registers	2	18
	Data format	2	20
	Short byte order signature	2	22
	Long byte order signature	4	24
Body	Body Data…	>=0	28

4.1.3 Buffer Type Identifiers

Name	Integer Identifier
DATABF	1
SCALERBF	2
SNAPSCBF	3
STATEVARBF	4
RUNVARBF	5

PKTDOCBF	6
BEGRUNBF	11
ENDRUNBF	12
PAUSEBF	13
RESUMEBF	14
PARAMDESCRIP (unused)	30

4.1.4 Predefined Body Layouts for Specific Buffer Types

As was already mentioned, the body portion of the 8k buffer consists of a single or many contiguous data structures. These are defined below and are associated with the type identifier.

4.1.4.1 Control Data (applicable types: BEGRUNBF, ENDRUNBF, PAUSEBF, RESUMEBF)

Description	Size (bytes)
Title	80
Time since run start	4
Time – month	2
Time – day	2
Time – year	2
Time – hours	2
Time – minute	2
Time – second	2
Time – tenth of second	2

4.1.4.2 Scaler Data (applicable types: SCALERBF, SNAPSCLBF)

Description	Size (bytes)
Interval offset end	4
UNDEFINED	6
Interval offset begin	4
UNDEFINED	6
Scaler values	4*(number of events)

4.1.4.3 Physics Event (applicable types: DATABF)

Description	Size (bytes)
Inclusive size	2
Generic data	>=0

4.1.4.4 Text Data (applicable types: RUNVARBF, PKTDOCBF, STATEVARBF)

Description	Size (bytes)
Inclusive size	2
Null-terminated strings	>=0

4.1.5 Functional software definitions – C++

If the user is interested in viewing software representations of what has been described, they can be found in buffer.h and buftypes.h of any 10.0 NSCLDAQ source tree.

4.2 VERSION 10.0

4.2.1 Overview

In the version 10.0 format, every complete entity passed between processes is called a ring item. Ring items differ from 8k buffers substantially but most obviously because they are variable length entities. Ring items consist of a header section followed immediately by a body. The header consists of two 32-bit integers identifying the total size of the ring item and the type of content it stores in the body. The content of the body is specific to one event and has either a predefined or an undefined structure depending on the type.

The most commonly sized data element in the ring item is the 32-bit integer. Byte ordering information is actually encoded in the type identifier. The type identifier is actually a 16-bit integer stored in a 32-bit integer. Since the top 16 bits of this integer must be zero, byte ordering can be inferred by noting which 16-bit field of that 32-bit integer is non-zero.

4.2.2 Top-level Memory Layout – "Ring Item"

	Description	Size (bytes)
Header	Inclusive size	4
	Type	4
Body	Data...	>= 0

4.2.3 Ring Item Types

Name	Value
BEGIN_RUN	1
END_RUN	2
PAUSE_RUN	3
RESUME_RUN	4
PACKET_TYPES	10
MONITORED_VARIABLES	11
INCREMENTAL_SCALERS	20
TIMESTAMPED_NONINCR_SCALERS	21
PHYSICS_EVENT	30
PHYSICS_EVENT_COUNT	31
EVB_FRAGMENT	40
EVB_UNKNOWN_PAYLOAD	41
FIRST_USER_ITEM_CODE	32768

4.2.4 Predefined Body Layouts for Specific Ring Items

The ring item header is followed immediately by a body whose structure is potentially predefined. These structures are defined below.

4.2.4.1 State Transition (applicable types: BEGIN_RUN, END_RUN, PAUSE_RUN, RESUME_RUN)

Description	Size (bytes)
Run number	4
Time offset	4
Timestamp (Unix)	4
Title	80

4.2.4.2 *Incremental Scaler (applicable types: INCREMENTAL_SCALER)*

Description	Size (bytes)
Interval Start Offset	4
Interval End Offset	4
Timestamp (Unix)	4
Scaler count	4
Scaler values	4*(Scaler count)

4.2.4.3 *Timestamped Scalers (applicable types: TIMESTAMPED_NONINCR_SCALER)*

Description	Size (bytes)
Event timestamp	8
Interval Start Offset	4
Interval End Offset	4
Interval divisor	4
Timestamp (Unix)	4
Scaler count	4
Scaler values	4*(Scaler count)

4.2.4.4 *Text (applicable types: PACKET_TYPES, MONITORED_VARIABLES)*

Description	Size (bytes)
Time offset	4
Timestamp (Unix)	4
String count	4
Null-terminated strings	>=0

4.2.4.5 *Physics Event (applicable types: PHYSICS_EVENT)*

Description	Size (bytes)
Data	>=0

4.2.4.6 *Physics Event Statistics (applicable types: PHYSICS_EVENT_COUNT)*

Description	Size (bytes)
Time offset	4
Timestamp (Unix)	4
Event count	8

4.2.4.7 *Event Builder Fragment (applicable types: EVB_FRAGMENT, EVB_UNKNOWN_PAYLOAD)*

Description	Size (bytes)
Event timestamp	8
Source id	4
Payload size	4
Barrier type	4
Payload	>=0

4.2.5 Functional software definitions – C++

If the user is interested in viewing software representations of what has been described, they can be found in DataFormat.h of any 10.0 NSCLDAQ source tree.

4.3 VERSION 11.0

4.3.1 Overview

The reader should refer to the overview of the version 10.0 data format to understand the 11.0 format, because it is the same. The main difference between 10.x and 11.x ring items is the ability to add a body header that contains event building information. If this header is not present, the body header is replaced by a zero.

4.3.2 Memory Layout

The basic data layout is as follows. The order of the data is guaranteed as well as the size of each data element. It is also guaranteed that no padding bytes exist between each element listed.

With no Body Header

	Description	Size (bytes)
Header	Inclusive size	4
	Type	4
Body Header	Size = 0	4
Body	Data…	>=0

Figure 1 Memory layout of NSCLDAQ 11.0 Format without Body Header

	Description	Size (bytes)
Header	Inclusive size	4
	Type	4
Body Header	Size = 20	4
	Timestamp	8
	Source id	4
	Barrier type	4
Body	Data…	>=0

Figure 2 Memory layout of NSCLDAQ 11.0 Data Format with Body Header Present

4.3.3 Types of Ring Items in NSCLDAQ 11.0

Name	Integer Identifier
BEGIN_RUN	1
END_RUN	2
PAUSE_RUN	3
RESUME_RUN	4
ABNORMAL_ENDRUN	5
PACKET_TYPES	10
MONITORED_VARIABLES	11
RING_FORMAT	12
PERIODIC_SCALERS	20
PHYSICS_EVENT	30
PHYSICS_EVENT_COUNT	31
EVB_FRAGMENT	40
EVB_UNKNOWN_PAYLOAD	41
EVB_GLOM_INFO	42
FIRST_USER_ITEM_CODE	32768

4.3.4 Predefined Body Layouts for Specific Ring Items

The ring item header is followed immediately by the optional body header section and then by a body whose content structure may be predefined. The predefined structures are detailed in the remainder of this section.

4.3.4.1 State Transition Body (applicable types: BEGIN_RUN, END_RUN, PAUSE_RUN, RESUME_RUN)

- Body header may or may not be present.

Description	Size (bytes)
Run number	4
Time offset	4
Timestamp (Unix)	4
Offset divisor	4
Title	80

4.3.4.2 Scaler Body (applicable types: PERIODIC_SCALERS)

- Body header may or may not be present.

Description	Size (bytes)
Interval Start Offset	4
Interval End Offset	4
Timestamp (Unix)	4
Interval divisor	4
Scaler count	4
Is Incremental	4
Scaler values	4*(Scaler count)

4.3.4.3 Text Body (applicable types: PACKET_TYPES, MONITORED_VARIABLES)

- Body header may or may not be present.

Description	Size (bytes)
Time offset	4
Timestamp (Unix)	4
String count	4
Offset divisor	4
Null-terminated strings	>=0

4.3.4.4 Physics Event (applicable types: PHYSICS_EVENT)

- Body header may or may not be present.

Description	Size (bytes)
Data	>=0

4.3.4.5 Physics Event Statistics (applicable types: PHYSICS_EVENT_COUNT)

- Body header may or may not be present

Description	Size (bytes)
Time offset	4
Offset divisor	4
Timestamp (Unix)	4
Event count	8

4.3.4.6 *Data Format (applicable types: RING_FORMAT)*

- **Never has a body header**

Description	Size (bytes)
Major version	2
Minor version	2

4.3.4.7 *Event Builder Fragment (applicable types: EVB_FRAGMENT, EVB_UNKNOWN_PAYLOAD)*

- **Always has a body header**

Description	Size (bytes)
Payload	>=0

4.3.4.8 *Event Building Parameters (applicable types: EVB_GLOM_INFO)*

- **Never has a body header**

Description	Size (bytes)
Coincident ticks	8
Is Building?	2
Timestamp Policy	2

4.3.5 Functional software definitions – C++

If the user is interested in the details of the software implementation details in this section, they can be found in DataFormat.h of any 11.0 NSCLDAQ source tree.

5 Consequences of Converting From Version 8.0:

NOTE: At the moment we already provide a conversion from 8.0 directly to 11.0 format. There is no support for converting between 8.0 and 10.0.

5.1 Converting to 10.0:

The contents of the 8k buffer header are mostly discarded, because the 10.0 format does not provide any support for many of the elements. This is because in some cases a single 8.0 buffer results in more than one 10.0 ring item. **The following pieces of information are always discarded unless explicitly described as being included in a ring item body**:

- Run number

- Buffer checksum (its value is no longer valid)
- Buffer sequence number
- Number of events
- Number of LAM mask in buffer (no longer used in 8.0 anyway)
- CPU processor number (no longer used in 8.0 anyway)
- Number of bit registers (no longer used in 8.0 anyway)
- Data format
- Short byte order signature
- Long byte order signature

The transformations made on the 8k buffer to produce version-10.0 ring items are:

5.1.1 DATABF:

- Type will map as DATABF → PHYSICS_EVENT
- Data buffers (DATABF) actually contain multiple physics event bodies within them. The original DATABF buffer will therefore be split into multiple ring items of type PHYSICS_EVENT. The size in each ring item produced will be computed according to the inclusive body header in the original physics event body. The total number of ring items produced will be the same as "number of events" in the 8k buffer header.

5.1.2 SCALERBF

- Size of the ring item will be computed
- Type will map as SCALERBF→INCREMENTAL_SCALERS (Note that this is not guaranteed to be accurate for the type. It could have been better suited for TIMESTAMPED_NONINCR_SCALERS, but there is no information to make this decision.)
- "Interval offset end" value used for "interval end offset".
- "Interval offset begin" value used for "interval start offset".
- Timestamp is filled in using the value returned by the std::time(std::time_t*) function in C++ at the time of conversion. (Note that this has no meaning to the data but serves to produce a functional value).
- The "number of events" in buffer header is used to set "scaler count".
- Scaler data transforms unchanged (both formats store value in 32-bit integers).

5.1.3 STATEVARBF, RUNVARBF, and PKTDOCBF:

- Size of the ring item will be computed based on the number of strings and their collective length.
- The data types map as:
  - STATEVARBF → MONITORED_VARIABLES
  - RUNVARBF → MONITORED_VARIABLES
  - PKTDOCBF → PACKET_TYPES
- Time offset will be set to 0.
- Timestamp (Unix) will be set to the value returned by the C++ std::time(std::time_t*) at time of conversion.
- The "number of events" becomes the "string count" value

- String content will remain the same, though an uninitialized padding byte following a null terminator may be added *between* strings. The padding byte serves the purpose of aligning the start of every string to an even address.

5.1.4 BEGRUNBF, ENDRUNBF, PAUSEBF, and RESUMEBF

- Size of the ring item will be computed based on size of body.
- The type of 8k buffer will be translated to a ring item type as follows:
    - BEGRUNBF → BEGIN_RUN
    - ENDRUNBF → END_RUN
    - PAUSEBF → PAUSE_RUN
    - RESUMEBF → RESUME_RUN
- Run number from buffer header will be used to set the run number in the new ring item.
- "Time since start of run" will transform to the "time offset".
- Time values for year, month, day, hour, min, second, and tenth of second are translated into unix time and used as the timestamp. *The tenths of second are discarded during this conversion.*
- Title will transform unchanged and will be null-terminated.

6 Consequences of Converting From 10.0

6.1 Converting to 8.0:

NOTE: We already provide a tool for converting 11.0 data format directly to the 8.0 format. The conversion below is largely based on what is implemented.

6.1.1 General rules for buffering:

The concept of buffering is largely tied to the protocol of the 8k buffer rather than the specific data elements contained in the buffer. It is therefore left undefined whether any attempt will be made to consolidate multiple ring items of the same type into a single 8k buffer.

The size of the fixed 8.0 buffer produced by the transformation will be specified at run time. The default value of 8192 bytes will be assumed. If it is impossible for a conversion to succeed meaningfully because of the size of the buffer. The conversion will fail and the user may have to restart the conversion software with a larger buffer size. At the most extreme scenario in which a ring item contains more than 128 kilobytes of data of invariant nature, the conversion may fail completely. The 8.0 format simply is incapable of representing such kinds of data.

6.1.2 BEGIN_RUN, END_RUN, PAUSE_RUN, and RESUME_RUN

- The content size will be computed based on the size of the ring item.
- The types will map accordingly:
    - BEGIN_RUN → BEGRUNBF
    - END_RUN → ENDRUNBF
    - PAUSE_RUN → PAUSEBF
    - RESUME_RUN → RESUMEBF
- The checksum will be set to 0.

- The run number will be used from the ring item and stored to produce 8.0 buffers until the next state change item is converted.
- The sequence number will be computed according to the PHYSICS_EVENT_COUNT and the number of previously converted PHYSICS_EVENT ring items.
- The number of events in buffer will be set to 0.
- The processor number will be set to 0.
- The number of bit registers will be set to 0.
- The data format will be set to 5.
- The short byte order signature and long byte order signature will be selected to match the byte ordering of the original ring item.
- The elements of the "state transition item" will map to a "control body" in the in the 8k buffer.
    - The first 79 characters of the title will transformed unchanged while the 80th will be replaced by a null terminator.
    - "Time offset" will be used to set "time since run start".
    - Timestamp (unix) will be unpacked appropriately into month, day, year, hour, minute, and second. The tenths place will be set to 0.
    - Offset divisor will be discarded.

6.1.3 PACKET_TYPES and MONITORED_VARIABLES

- The content size will be computed based on the size of the ring item.
- The types will map as follows:
    - PACKET_TYPES → PKTDOCBF
    - MONITORED_VARIABLES → RUNVARBF
- The checksum will be set to 0.
- The run number will be set to 0 by default. If it can be determined by data converted beforehand, the previously extracted run number will be used.
- The sequence number will be computed according to the PHYSICS_EVENT_COUNT and the number of previously converted PHYSICS_EVENT ring items.
- The number of events in buffer will be set to N, where N is the total number of contiguous null-terminated strings that will fit into a single 8k buffer.
- The processor number will be set to 0.
- The number of bit registers will be set to 0.
- The data format will be set to 5.
- The short byte order signature and long byte order signature will be set to 0x0102 and 0x01020304 respectively.
- The null-terminated strings will be copied into a new 8k buffer unaltered. They will be null-terminated and a padding byte may be added in between them in order to aligned them on even memory boundaries. If the total number of strings cannot fit, multiple 8k buffers may be sent.
- ***An error will occur if a null-terminated string will not fit into the 8.0 format buffer. The conversion process will cease and the user will need to initiate the conversion using a larger buffer size.***

6.1.4 PHYSICS_EVENT

- The content size will be computed based on the size of the ring item.

- The type will become DATABF
- The checksum will be set to 0.
- The run number will be set to 0 by default. If it can be determined by data converted beforehand, the previously extracted run number will be used.
- The sequence number will be computed according to the PHYSICS_EVENT_COUNT and the number of previously converted PHYSICS_EVENT ring items.
- The number of events in buffer will be set to N, where N is the total number of contiguous PHYSICS_EVENTs that will fit into a single 8k buffer.
- The processor number will be set to 0.
- The number of bit registers will be set to 0.
- The data format will be set to 5.
- The short byte order signature and long byte order signature will be set to 0x0102 and 0x01020304 respectively.
- The data body will be copied in to the next available space of the current 8k buffer and the size prepended to it will be computed from the size of the original ring item.
- ***An error will occur if the size of the ring item body is larger than the allotted storage of the 8.0 buffer. The conversion will cease and the use will have to start it over using a larger 8.0 buffer size.***

6.1.5 PHYSICS_EVENT_COUNT

- This will not result in an 8k buffer being produced. The data contained in the PHYSICS_EVENT_COUNT ring item will only be used for updating the sequence number. This is important for sampling applications.

6.1.6 EVB_FRAGMENT

- Will be discarded.

6.2 Converting to 11.0:

The effects will be considered on the basis of item type:

6.2.1 BEGIN_RUN, END_RUN, PAUSE_RUN, and RESUME_RUN:

- Size in ring item header will be recomputed.
- The types will map as:
  - BEGIN_RUN → BEGIN_RUN
  - END_RUN → END_RUN
  - PAUSE_RUN → PAUSE_RUN
  - RESUME_RUN → RESUME_RUN
- No body header will be included in the resulting 11.0 ring item.
- The run number will transform unchanged.
- The time offset will transform unchanged.
- The timestamp (Unix) will transform unchanged.
- The offset divisor will be set to 1.
- The title will transform unchanged.

6.2.2 PACKET_TYPES and MONITORED_VARIABLES:

- Size in ring item header will be recomputed.
- The type will transform as :
    - PACKET_TYPES → PACKET_TYPES
    - MONITORED_VARIABLES → MONITORED_VARIABLES
- No body header will be included in the 11.0 ring item.
- Time offset will transform unchanged.
- Timestamp (Unix) will transform unchanged.
- String count will transform unchanged.
- The offset divisor will be set to 1.
- The strings will transform unchanged.

6.2.3 INCREMENTAL_SCALERS:

- Size in ring item header will be recomputed.
- The type will map as INCREMENTAL_SCALERS → PERIODIC_SCALERS
- No body header will be included in the 11.0 ring item
- Interval start offset will remain the same
- Interval end offset will remain the same
- Timestamp (Unix) will remain the same
- Interval Divisor will be set to 1.
- Scaler Count will remain the same.
- “Is Incremental” will be set to 1.
- Scaler values will remain the same.

6.2.4 TIMESTAMPED_NONINCR_SCALERS:

- Size in ring item header will be recomputed.
- Type will map as TIMESTAMPED_NONINCR_SCALERS → PERIODIC_SCALERS
- No body header will be created in the 11.0 ring item.
- Interval start offset will remain the same
- Interval end offset will remain the same
- Interval divisor will be set to 1.
- Timestamp (Unix) will be become timestamp (unix).
- Scaler count will transform unchanged.
- “Is Incremental” will be set to 0.
- Scaler values will transform unchanged.
- Event timestamp will be discarded.

6.2.5 PHYSICS_EVENT:

- Size in ring item header will be recomputed.
- The type will map as PHYSICS_EVENT → PHYSICS_EVENT
- No body header will be created in the 11.0 ring item.
- Body data will transform unchanged.

6.2.6 PHYSICS_EVENT_COUNT:

- Size in ring item header will be recomputed.
- The type will map as PHYSICS_EVENT_COUNT → PHYSICS_EVENT_COUNT

- No body header will be included in the 11.0 ring item.
- Time offset will remain the same.
- Offset divisor will be set to 1.
- Timestamp (unix) will remain the same.
- Event count will remain the same.

6.2.7 EVB_FRAGMENT

- Size in ring item header will be recomputed.
- Type will map as EVB_FRAGMENT → EVB_FRAGMENT
- Body header will be created in the 11.0 ring item.
  - Body timestamp will be used for body header timestamp
  - Body source id will be used for body header source id
  - Payload size will be discarded. The value can easily be calculated from the size value in the RingItemHeader and the size of the body header.
  - Body barrier type will be used for body header barrier type
- Payload will remain the same. Note that the payload may consist of fragments in a given format and this is not attempting to convert them. The result will be that the top level format of the built ring item will correspond to version 11.0, but the payload will contain fragments in the 10.0 format.

7 CONSEQUENCES OF CONVERTING FROM 11.0

7.1 CONVERTING FROM 11.0 TO 10.0

In general, the conversion from version 11.0 to version 10.0 is minimal outside of the body header.

7.1.1 State Transition Items (BEGIN_RUN, END_RUN, PAUSE_RUN, RESUME_RUN)

- Size in ring item header will be recomputed.
- Types will transform as:
  - BEGIN_RUN → BEGIN_RUN
  - END_RUN → END_RUN
  - PAUSE_RUN → PAUSE_RUN
  - RESUME_RUN → RESUME_RUN
- The body header will be discarded if it exists.
- Run number will remain the same.
- Time offset will remain the same.
- Timestamp (unix) will remain the same.
- Offset divisor will be discarded.
- Title will remain the same.

7.1.2 PERIODIC_SCALERS

If PERIODIC_SCALERS is NOT increment:

- Size in ring item header will be recomputed.

- Type will map as PERIODIC_SCALERS → TIMESTAMPED_NONINCR_SCALER
- If a body header is present, only the timestamp will be salvaged and set to the "event timestamp". All other body header information will be discarded.
- The interval start offset will transform unchanged.
- The interval end offset will transform unchanged.
- The timestamp (unix) will be set to the "clock timestamp"
- The interval divisor will transform unchanged.
- The scaler count will transform unchanged.
- The "is incremental" value will be discarded.
- The scaler values will transform unchanged.

If PERIODIC_SCALERS is incremental:

- Size in ring item header will be recomputed.
- Type will map as PERIODIC_SCALERS →INCREMENTAL_SCALER
- The body header will be discarded if present.
- The interval start offset will transform unchanged.
- The interval end offset will transform unchanged.
- The timestamp (unix) will transform unchanged.
- The interval divisor will be discarded.
- The scaler count will transform unchanged.
- The "is incremental" value will be discarded.
- The scaler values will transform unchanged.

7.1.3 PACKET_TYPES and MONITORED_VARIABLES

- Size in ring item header will be recomputed.
- Type will map as:
  - PACKET_TYPES → PACKET_TYPES
  - MONITORED_VARIABLES → MONITORED_VARIABLES
- Any body header will be discarded if present.
- Time offset will transform unchanged.
- Timestamp (Unix) will transform unchanged.
- String count will transform unchanged.
- Strings will transform unchanged.

7.1.4 PHYSICS_EVENT_COUNT

- Size in ring item header will be recomputed.
- Type will map as PHYSICS_EVENT_COUNT → PHYSICS_EVENT_COUNT
- Body header will be discarded if present.
- Time offset will transform unchanged.
- Offset divisor will be discarded.
- Timestamp (Unix) will transform unchanged.
- Event count will transform unchanged.

7.1.5 PHYSICS_EVENT

- Size in ring item header will be recomputed.

- Type will map as PHYSICS_EVENT → PHYSICS_EVENT
- Body header will be discarded if present.
- All data in the body will transform unchanged.

7.1.6 RING_FORMAT

- This will be discarded.

7.1.7 EVB_FRAGMENT

- Size in ring item header will be recomputed
- Type will map as EVB_FRAGMENT → EVB_FRAGMENT
- Body header timestamp will become timestamp in body
- Body header source id will becomes source id in body
- Body header barrier type will become barrier type in body
- Payload size will be computed based on size of ring item.
- Payload will be transformed unchanged.

7.1.8 EVB_UNKNOWN_PAYLOAD

- The transformation rules are the same as for EVB_FRAGMENT.

7.1.9 EVB_GLOM_INFO

- This will be discarded.

8 CONCLUDING REMARKS

At this point, the reader should have a solid handle on what is involved when converting between data formats. Some final remarks follow in light of the above transformations:

1. **NSCLDAQ-provided utilities for setting up the data stream will support streaming of data in a single format that is consistent with the version of NSCLDAQ in use.**
   Transforming data between formats should not be done lightly, because it can easily discard information taken during an experiment. For this reason, if users want to convert the data to a certain format, it can always be accomplished as a post-processing step or a step immediately before an analysis code accesses it.
2. Reversible transformations can be constructed provided knowledge of certain external logic (e.g. experiment-specific code). Any person who knows this logic and desires to construct such transformations for a specific set of data is free to do so. The contents of this document should provide ample detail to succeed.
3. If a user seeks to prevent changing an analysis code between versions, they would do well to write it in way that is agnostic to data format. By doing so, they could simply call this code in separate software that understands how to traverse the specific data format and call the analysis code at the appropriate point. Software constructs for working with each data format is implemented and available for use, so writing this separate software is in general straightforward.