L2STT Unpacking

Harrison B. Prosper
STT Group

Abstract
Here are some details of the proposed design of code to unpack data from L2STT.

1 INTRODUCTION

The Silicon Track Trigger produces data at Level 2, which are forwarded to Level 3 where they are packed into the RawDataChunk. The purpose of the STT unpacker is to unpack these data from the RawDataChunk. By unpacking I mean transforming the un-structured (but ordered) data in the RawDataChunk into structured data represented as objects. The unpacking of data from the RawDataChunk is sometimes referred to as *Level 3 unpacking*.

1.1 Crate Identifiers and Modules

In a RawDataChunk data are structured into blocks called *modules*. Each module consists of an array of 32-bit words preceded by a 32-bit word containing the word count. A module, which corresponds to the data associated with a given VME address in a given crate, is identified by its *crate identifier* and its *module number*. The module number is the *ordinal* value of the module within the data block pertaining to a crate, that is, it is the order in which cards are read out. For the STT the crate identifiers range from 0x70 to 0x75, while the module numbers range from 0 to 11.

1.2 Goals

The goal of the unpacker is to create objects that expose *useful* structure in the data within modules. For some applications, such as Examine, it is useful, indeed necessary, to expose all relevant fields. It is also convenient, and good practice, to shift the burden of doing so from the user to the objects. The STC group, however, requires great flexibility and needs objects that leave the data uninterpreted. The interpretation is to be handled by code external to the objects.

2 DESIGN

The STT system is moderately complicated in that it contains more than two dozen distinct bit-patterns. Therefore, to expose all relevant fields requires several dozen methods in addition to those that leave the fields unexposed. These methods are divided amongst classes that model blocks of data within modules. Below is a tentative list of classes that have been identified to model these blocks. Each class is a standard C++ class with methods for setting and getting attributes. Complicated classes are built by containment, that is, they have components that are objects. The symbol [] indicates "an array of" and the indentation indicates containment. For the STC, the objects that leave the data uninterpreted are the same as those that expose the fields except that the arrays of objects are replaced by arrays of unsigned integers.

2.1 Classes

```
Fiber Road Card
---------------
          STTFRCCard                    Exposes all fields
                STTFRCL3Header
```

```
                STTFRCHeader
                STTCTT[]                Either 1 or 0 block depending on
                        L2Header        whether we have a full event or not
                        STTCTTTrack[]
                        L2Trailer
                        pad[]
                STTFRCTrailer
                STTFRCL3Trailer

Silicon Trigger Card
--------------------
        STTSTCCard                      We interpret the header
                STTSTCL3Header          and status words only, which
                STTSTCStatus[8]         are used to decode the remaining
                                        blocks, either fully or not at all.
        STTSTCL3Trailer

        STTRoadBuf
                STTRoadHeader
                STTFRCHeader
                L2Header
                STTCTTTrack[]           Since this is stable, we expose all
                L2Trailer               fields
                STTRoadTrailer

        STTHitBuf
                STTHitHeader
                STTHit[]                Expose fields + simple array
                STTHitTrailer

        STTZCentroidBuf
                STTZCentroidHeader
                STTZCentroid[]          Expose fields + simple array
                STTZCentroidTrailer

        STTACentroidBuf
                STTACentroidHeader
                STTACentroid[]          Expose fields + simple array
                STTACentroidTrailer

        STTClusterBuf
                STTClusterHeader
                STTCluster[]            Expose fields + simple array
                        STTStrip[]
                STTClusterTrailer

        STTRawBuf
                STTRawHeader
                STTRaw[]
                        STTRawSeqHdi
```

```
                    STTRawChipId
                    STTRawOther
            STTRawTrailer

        STTBadBuf
            STTBadHeader
            STTBad[]
            STTBadTrailer

Track Trigger Card
------------------
        STTTFCCard                      Number of blocks can vary.
            STTTFCL3Header

        Optionally an FRC block
        Optionally an STC block

        STTTFCTrackBuf                  Same object as sent to L2Global
            L2Header
            L2STTTrack[]
            L2Trailer

        STTTFCL3Trailer
```

2.2 Unpacking algorithm

The algorithm is straightforward. It assumes that the data are self-describing and that the unpacker is given the identifier of the crate to be unpacked. Here is the proposed algorithm as *pseudo-code*.

```
With crate-id

For each module number in (0 ... 11)
   If module exists Then
      Extract module from RawDataChunk
      Unpack module header and determine its identity

      If     source is FRC Then
         Unpack module into STTFRCard      (A)

      ElseIf source is STC Then

         Unpack module into STTSTCCard     (B)
         For each STTSTCStatus word
            If     blockType is ROAD
               Unpack STTRoadBuf
            ElseIf blockType is HIT
               Unpack STTHitBuf
            ElseIf blockType is ZCENTROID
               Unpack STTZCentroidBuf
            ElseIf blockType is ACENTROID
```

```
                  Unpack STTACentroidBuf
               ElseIf blockType is CLUSTER
                  Unpack STTClusterBuf
               ElseIf blockType is RAW
                  Unpack STTRawBuf
               ElseIf blockType is BAD
                  Unpack STTBadBuf
               EndIf
            EndFor
            Unpack STTSTCL3Trailer

         ElseIf source is TFC Then
            Unpack module into STTTFCCard     (C)

         EndIf
      EndIf
   EndFor
```

The pseudo-code contains 3 unpacking blocks (A), (B) and (C). If the I/Ogen mechanism is used for unpacking, all objects and their associated interpretation code is generated automatically from a data definition file, l2stt.iogen that resides in the l2io package. The unpacking of each block type, FRC, STC and TFC, would be effected by a call to the *retrieve* method of a *DataBroadcast* object, which models a module. A DataBroadcast object is a *smart buffer* that understands how to unpack blocks of data into I/Ogen objects. This works because I/Ogen objects follow a uniform unpacking protocol.

If the desired object has a fully-specified structure, a single call to the retrieve method of the DataBroadcast object is sufficient to recursively unpack the relevant block of data. This is what would happen in code block (A). Moreover, if the *order* of blocks is fixed but there is a possibility that blocks may be missing, the retrieve method is able to handle this situation provided that the data are self-describing. This is the case for the FRC and, I presume, for the TFC data blocks, which are unpacked using code blocks (A) and (C), respectively. If the *order* of blocks is not fixed, as is the case for STC data blocks, the retrieve method cannot be used to fully, and recursively, unpack its associated module. Instead it is necessary to use the *retrieve2* method of the DataBroadcast object to unpack block by block, where the identity of the object into which a block is to be unpacked, at each stage, is made by the unpacker as in code block (B).

3 WHY I/OGEN?

I/Ogen is based on the following observation: Unpacking is a routine algorithmic mechanism, which entails imposing structure on un-structured data. Therefore, given a data definition file that describes the structure to be imposed, unpacking can in principle be automated. I/Ogen is a useful step in that direction and follows an important trend in computer science in which data definition languages (such as XML) are used to describe data, in as much detail as desired, so that they can be acted upon by automated systems. Obviously this approach is powerful. However, mere opinion is an insufficient basis to use I/Ogen. I have more prosaic reasons:

- It is mandated by the trigger simulator group for the trigger simulator.
- It avoids the need to write an STT-specific scheme to inject objects into the framework's dataflow system.
- It avoids the need to write an STT-specific RawDataChunk packer.

- It allows the extraction of vectors of (pointers to) objects of a given type from the RawDataChunk using a simple interface: A call to the template function extract().
- It hides all interpretation code inside objects where good practice suggests it should be.
- It avoids the need to hand-code *routine* unpacking classes, such as those listed above.
- It imposes *no* restriction on the desired level of interpretation of the un-structured data. Moreover, the data definition language, which is DØ-specific—it could have been XML, is simple to understand. Any change in the data definition of an object triggers automatic re-generation of the affected code.

Presumably, to be useful in a computer program, such as Examine, data have to be interpreted by *some* code. Therefore, almost any change in interpretation of data, usually because of hardware or firmware changes, entails recompilation and relinking of the interpretation code whether that code is written by hand or by a code generator. Even though I have chosen, for the reasons given above, to use generated code this choice does *not* preclude the provision of generated code that leaves the interpretion of data to hand-written code.