Version 2.0: October 7, 1998
Version 1.0: February 17, 1998

STAR Note 325
Structures for Distribution and Analysis of STAR TPC Data ("Second Iteration")

R. Bossingham
Nuclear Science Division
Lawrence Berkeley National Laboratory

A format, involving STAF tables within a directory structure, is defined for STAR TPC raw data distribution and analysis. Normally, the TPC data are mapped onto the TPC pad planes, but a secondary format is based on the front-end electronics.

The relationship of the data structures to the on-line event server and off-line cluster/hit-finding is explored, since both should be able to use the structures efficiently. A weak restriction is that sequence lengths cannot exceed 256 time bins.

The format evolved from, but is not compatible with, the "MiniDAQ" cosmic-ray test format.[1] The new format better supports 24 TPC sectors, indexes the data at finer granularity, and is more appropriate for event serving, use with the real DAQ, and high track density. **Venus+tss** simulation results[2] predict an overhead of ~21% for central Au-Au events. The ~61 KB/event overhead for sparse events (e.g., peripheral collisions) is higher percentage-wise (~285%), but still acceptable in absolute terms.

Introduction

STAR TPC data is to be transmitted and recorded in "DAQ" format, which is matched to the DAQ hardware and, due to compression, may be opaque until translated by DAQ-supplied library routines. Translation into a TPC analysis format would occur for both online event serving and offline for events read from tape; the analysis format must be appropriate for the event server and diagnostics, as well as for the off-line analysis. It must also reasonably support the full range of TPC data densities, from single cosmic rays to central Au-Au collisions and pedestal events.

The structures proposed in earlier versions of SN0325 elicited comments, corrections, complaints, questions, suggestions, references and other information; other discussions predated SN0325. The current design accommodates these to the extent possible. I am indebted to Iwona Sakrejda, Doug Olson, Mike LeVine, Kathy Turner, John Nelson, Mark Gilkes, Nathan Stone, Susanna Jacobson, Craig Tull, Lanny Ray, Claude Pruneau, Jo Schambach, Bill Greiman and others.

[1] R. Bossingham, STAR Note SN0282 (1997).

[2] Iwona Sakrejda, http://www.rhic.bnl.gov/star/starlib/doc/www/html/tpc_l/tpc.html .

Purpose of Structures

The TPC raw data structures are defined, first and foremost, for cluster and hit finding, replacing the inefficient and restrictive[3] tppad/tppixel structures. As such, the primary format relates to the TPC's physical subdivisions—sector, pad row, pad and time bucket. The analysis software (e.g., **tcl**) will, of course, have to be changed to exploit the new format fully.

For simplicity, a single format should represent the raw data after it is translated from DAQ format and moved through event distribution, and into on-line data displays, or cluster and hit finding. Also, until the Monte Carlo chain supports translation into, and out of, DAQ format, the structures will replace tppad/tppixel for Monte Carlo data storage. A secondary format, based on the electronics subdivisions—sector, readout board, Front-End Electronics (FEE) card, channel and time bucket—is defined to be used for electronics study and debugging in atypical, dedicated runs, accessing "blind" channels.

Neither format directly supports simple, visual inspection of the data. Such tools (e.g., the existing XYZ format) are needed, but a non-hierarchical format like XYZ is incompatible with efficient storage, distribution and analysis.

TPC Subdivisions

The information associated with the TPC subdivisions affects the format design and includes:

Sector, subsector: ID; geometrical position; data availability; time-bin range.

Pad row: ID, pads with data entries, row location and number of pads in row. Cluster and hit finding in **tcl** is by pad row, and calibrations could appear at this level (ground, anode and field wires span a pad row).

Readout board (RDO): ID; RDO status via Slow Controls; FEE location list; FEE cards with data entries; trigger-time offset.

FEE card: ID on RDO and ID on sector; channels with data.

Pad/channel: ID; channel status; t_0 calibration; time buckets with data entries; FEE pulser status; fixed-length lists of ADC, pedestal, or noise data.

Time bucket: ID; ADC value (with/without pedestal subtraction or gain correction; linear, non-linear or truncated; 8- or 10-bit, or floating); pedestal mean; rms; drift of pedestal mean.

[3] Inefficient because they require bit-level manipulation and have $\sim$325% storage overhead for dense data; restrictive because only 10-bit ADC values are supported.

General Considerations

Practical considerations shaping the format include:

- Must support up to 24 sectors of two subsectors and 45 pad rows with up to 182 pads/row and 512 time buckets/pad; or 6 readout boards with up to 36 FEE cards and 32 channels/FEE card.
- A fixed number of time buckets (e.g., 512) *cannot* be assumed; the ADC sampling frequency may be reduced, subdividing the TPC drift volume into fewer time buckets (e.g., ~300).
- Anode-wire data will be recorded by Trigger and need *not* be supported.
- The primary purpose is not bulk storage; convenience and speed should be emphasized.
- Generalized structures are needed for the event server to give all data consumers subsets of the data in a similar form.
- Data subsets at the supported granularity must be efficiently extractable by the event server.
- The cluster and hit finders must access the data at the subsector, pad row and pad levels.
- Structures should be compatible with the others associated with the cluster and hit finder.
- Efficient, direct referencing of a single data table by both pads and FEE channels is precluded by the random pad-to-channel mapping; but indirect referencing can, and must, be supported.
- The full range of STAR data densities must be supported; flexibility, based on simple criteria, is desireable, both event by event and within an event.
- Sector-level directories of parallel structure should hold tables for first-pass cluster- and hit-finding.
- Clusters (and, therefore, pad rows) should not be split across tables.
- Data should be grouped and ordered for efficient memory access.
- Table sizes should match the amount of data handled at one time; many small tables have excessive overhead; a few large tables tend to flush L2 memory caches.
- Table columns should hold a single data type; rows should map to a single entity.
- Explicit flags and values should be provided to simplify decoding.
- Decoding flags should appear above the table hierarchy level where they are applied, but not so high as to compromise generality unreasonably.
- Names should be mneumonic, consistent and extensible.
- Table length must be a multiple of the word length; data-type lengths cannot increase across a row; row lengths must be a multiple of their longest datum.

Event Server Granularity

Data structure issues and event server granularity are connected in that the structures must allow efficient data extraction at the required granularity; and, at fine granularities, data requests may resemble the data structures, since any structure that can represent a request for some set of pixels, can represent the same stored pixels, provided certain conventions are defined.)

Some obviously useful granularities for mapped data requests are full event, sector, subsector, pad row, pad and pixel. Other considerations, such as available event-distribution bandwidth, must be taken into account, of course. Assuming that hierarchical data structures are used to minimize overhead for dense data, finer granularity requires deeper access; pad-row granularity seems like a reasonable compromise.

DAQ Translator Output

TPC data will be output and recorded in a DAQ format to be translated online by the event server and offline as events are read from tape. Some comments, based on a draft of the DAQ format:

Overall form: Translation must fix endian problems and create Staf tables.

Contiguous sequences: Contiguous sequences should be joined at translation time, up to the 256 time-bin limit. Five bits suffices for the average, expected sequence length of 10, but will not make all useful sequences disjoint, and joining contiguous sequences in a later pass, or checking for such sequences in a sequence-based cluster finder, is expensive.

Huffman-encoded ADC's: DAQ optionally compresses ADC's as variable-length Huffman codes; these should be decoded once—at translation time. Variable-length codes preclude easy manipulation, sorting or random access, since all preceding codes for the mezzanine card (1/3 readout board) must be parsed first. The $\sim$13 SpecInt95-sec. required to decode a central Au-Au event is not dominant, but exceeds the cost of, say, cluster finding.

Other ADC operations: Operations, other than decompression, are of a different class. Pedestal subtraction, gain correction, and 8- to 10-bit conversion should be postponed until the analysis stage (the latter inflates the event size by a factor of two).

Data grouping: Data should be sorted and grouped by subsector during translation to avoid the inefficiency of either re-grouping it later or interposing mapping tables. (Pad-to-channel mapping can be convoluted, especially on the inner subsector, and further fragmentation occurs as readout board data are shared between three mezzanine cards.)

Readout board headers: The 64-byte readout-board headers that DAQ stores in the **TPCR-CLNDX** bank should be translated into tables in the corresponding sector's directory.

Clusters from the i960's: The clusters found by the i960's on one mezzanine card and stored in the **TPCMZCL** bank should be included in the translated event.

Nomenclature

Glossary

Black data: A dataset that includes every pixel in the allowed range.

DAQ: Data-AcQuisition—the system, or the group responsible for it.

FEE: Front-End Electronics; each card has 32 channels shared by two pairs of SAS/SCA chips.

Grey data: A dataset to which every pad/channel contributes, but without all allowed pixels.

Mapped: Mapped data structures relate to the physical TPC padplanes.

RDO: ReaDOut board. Reads and transmits digitized data from as many as 36 FEE cards.

SAS: STAR Amplifier/Shaper. Two SAS chips form each FEE card's front end.

SCA: Switched Capacitor Array. Two SCA chips digitize the data on each FEE card.

Sector : 30-degree sector on one side of the central membrane, each with an inner and outer subsector, and instrumented by six readout boards.

Sparse data: A dataset lacking entries for most pads/channels.

Unmapped (or native): Unmapped data structures relate to the TPC's underlying electronics.

Directory Naming

One must anticipate other subdetectors and other datasets, avoiding directory names that are either too simplistic (e.g., "DATA") or too narrow (e.g., "TPC_RAW_8_BIT_ADC_DATA"). Prepending all base-level TPC directory names with "TPC_" is an obvious convention, and names like "TPC_DATA" for raw and processed data, or "TPC_CAL" for calibration data, appear in figures in this note for illustration, but such conventions are separate from the format's substance.

Table Naming

Most tables contain several, qualitatively different information types, so names tend to be vague, incomplete or long. A convention is defined here that divides table names into three parts: *StructureType_RowEntity_Qualifier*. This provides short, descriptive table names, leaving enough flexibility to add new, similar tables for cluster and hit-finding, etc. *StructureType*'s include:

cls_ : Pixel clusters and related information from off-line analysis.
c96_ : Pixel clusters and related information from the DAQ i960's.
hit_ : Hits and related information.
raw_ : Data *structured* like raw ADC data (e.g., pedestal means or gains).
rcl_ : Offline cluster information relating to raw data only.

A *RowEntity* is described by one table row:

chn_ : **Chan**nel on a FEE card.

fee_ : Front-End Electronics (**FEE**) card; $\leq$36 per readout board.

grd_ : Grid element with Cartesian axes defined by one pad row and z.

pad_ : **Pad** within a pad row.

row_ : Pad **row** within a sector.

sec_ : Full **sector**.

seq_ : **Seq**uence of consecutive time bins on a pad (or channel).

Possible *Qualifier*'s include:

in : Inner subsector.

m : Mapped data.

out : Outer subsector.

ri : Readout board i; $i \in [1, 6]$.

u : Unmapped data.

Convention for Table Entries

Referencing Three options associate entities with rows in lower-level tables:

N : None; the lower-level table does not exist.

R : Range; ID's from the entire defined range are associated in order.

L : List; ID's are explicitly listed in a lower-level table.

S : Sequences; ID's are inferred from defined sequences.

Format Two basic formats are supported:

U : Unmapped or native. Data are referenced to readout boards and FEE cards.

M : Mapped. Data are referenced to the padplane.

Variable types Table entry types are defined according to the convention:

char : Octet, signed (or character).

u_char : Octet, unsigned.

short : Two-byte integer, signed.

u_short : Two-byte integer, unsigned.

long : Four-byte integer, signed.

u_long : Four-byte integer, unsigned.

Format Definition

Overview

The structures are defined with access, extraction, and storage efficiency in mind. They are intended to be "natural," exploiting STAF's capabilities—not just bags o'bytes. The format is, in general, weighted toward usability, but makes two major concessions to storage efficiency.

Two distinct formats—*mapped* and *unmapped*—are defined, along with flags to guide their interpretation. Pure data (usually at the pad or pixel level) are segregated into separate tables. Table 1 lists the basic tables for the raw data. Therefore, overhead is independent of the number of parallel data tables and adding or deleting data table requires no other changes. However, changing content *within* a data table can force significant changes, though this is mitigated by using index offsets, instead of absolute indices, when possible. In particular, it is expensive to interleave sets of tables, or to re-group and re-sort data.

Hierarchy	Mapped Data (*=**in**, **out**)	Unmapped Data (*=**1**,**2**,**3**,**4**,**5**,**6**)
Full TPC	**raw_sec_m**	**raw_sec_u**
Sector_N/ Level 1	**raw_row_***	**raw_fee_r***
Sector_N/ Level 2	**raw_pad_***	**raw_chn_r***
Sector_N/ Level 3	**raw_seq_***	**raw_seq_r***

Table 1: Raw data tables. Tables listed side by side are similar, with similar purposes.

Different TPC sub-directory branches can be created for different classes of information, such as calibration data and event data. A branch is largely described by a **raw_sec_m** table (or **raw_sec_u** for unmapped data). At the same hierarchy level, a sub-directory **Sector_N/** can be created for the data from each sector N. Tables parallel to **raw_sec_*** could sometimes be useful; the event server, for example, could mark which data tables exist. This approach should *not* be used in the formal analysis chain, since any defined list of tables will become too restrictive as the software develops.

Mapped Data Format

The mapped data structures are primarily intended for hit-, cluster- and track-finding. Figure 1 shows the basic scheme. The DAQ data stream is to contain the information needed to map data onto pads, without reference to external maps; such maps should not change during a run. Note that data from "blind" FEE channels (those unassociated with a pad) cannot be represented in a natural way. The semi-hierarchical table relationships for mapped data are shown schematically in Fig. 2 and, with more detail, in Fig. 3.

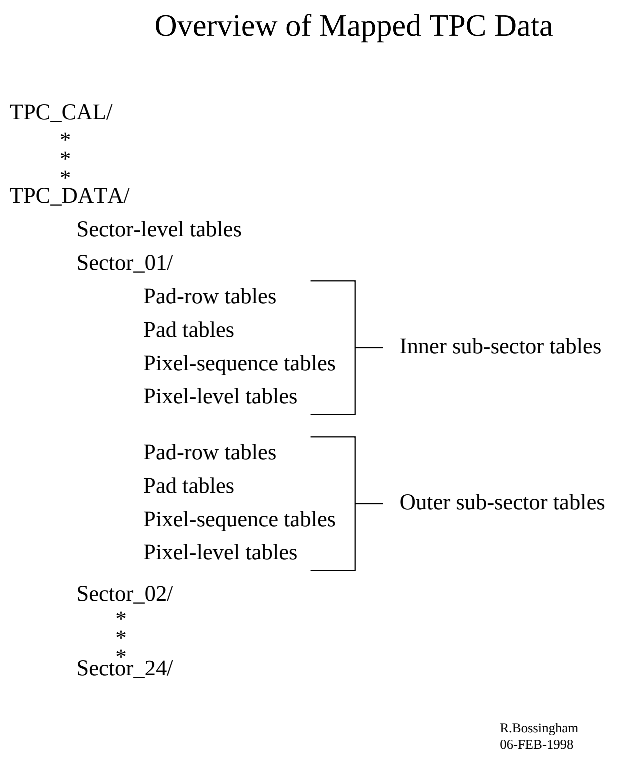

Figure 1: Overall mapped-data structure, showing TPC data and calibration branches. TPC_CAL and TPC_DATA have the same substructure.

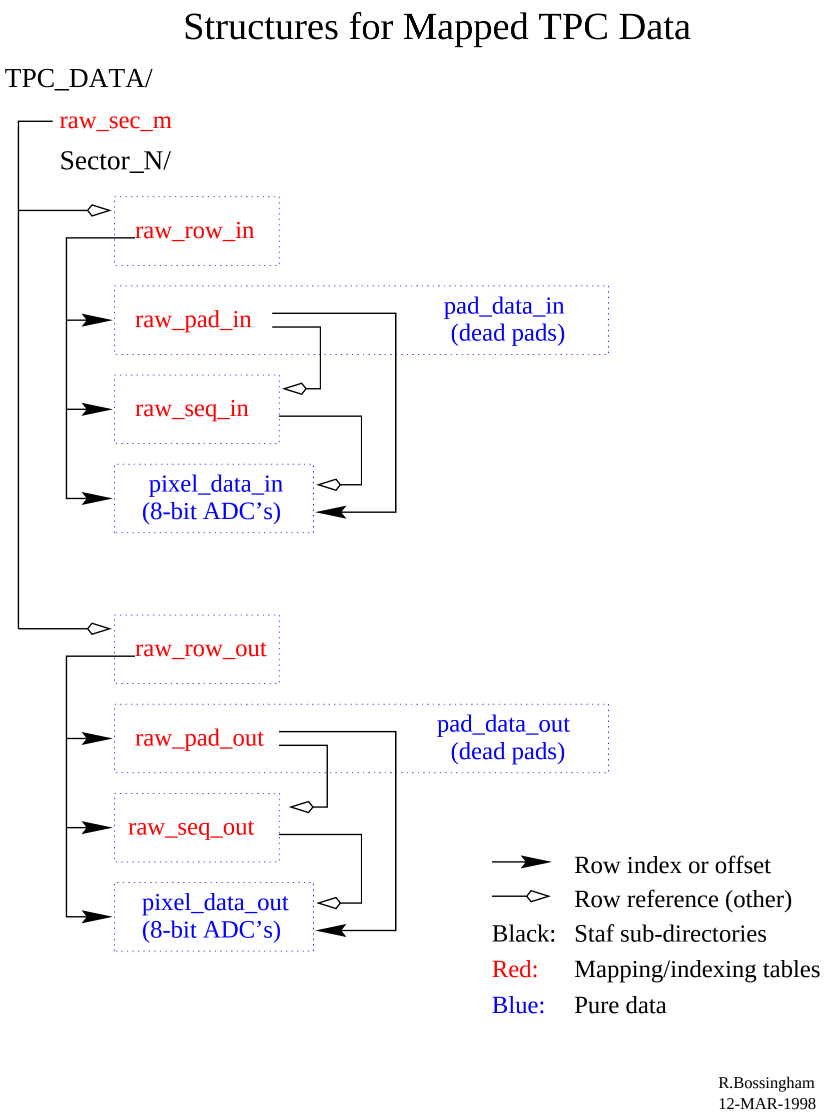

Figure 2: Structure for mapped TPC data. The overhead tables are discussed in the text; the data tables were chosen arbitrarily. The same essential structures also apply to other types of data (e.g., calibration data).

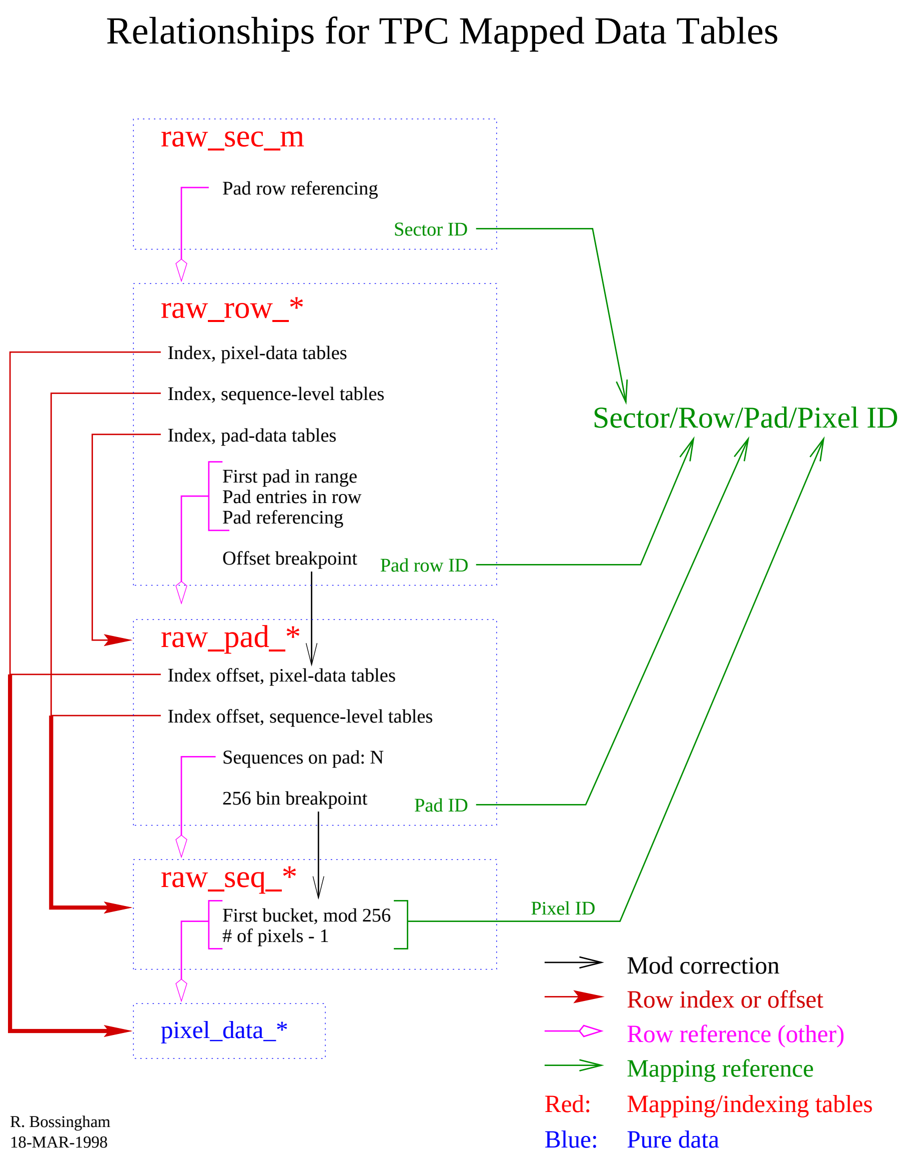

Figure 3: Table relationships and pixel referencing for mapped TPC data.

Sector Description

The top-level **raw_sec_m** table—see Table 2—has exactly 24 rows ordered by sector ID (1–24). Each row gives time-bin information and describes the pad-row tables by subsector. Its 192-byte size is negligible. Two implicit assumptions are:

1. Time-bin ranges and zero suppression are set at the sector level or above.
2. The method associating pad ID's with data might change below the sector level; choosing it at the pad row level may improve storage efficiency.

Column	Values	Type	Entity	Bytes
First time bin	1 – 512	u_short	Sector	2
Last time bin	1 – 512	u_short	”	2
Time-bin referencing	N, R, S	char	”	1
Pad-row referencing	N, R	char	Inner subsector	1
”	”	”	Outer subsector	1
Sector ID	1 – 24	u_char	Sector	1
Total bytes per row				8

Table 2: Description of one of the 24, eight-byte rows in the 192-byte **raw_sec_m** table. Rows are ordered by sector ID from 1–24 and define the existence and interpretation of subsequent tables.

Pad-Row Description

Pad-row data are described by the **raw_row_*** tables—see Table 3—giving absolute row indices into the pad, sequence and pixel tables. Giving the number of entries in the lower-level tables is redundant, but simplifies data extraction. The three options to reference the pads in subsequent tables are: 'N' for no entries, 'R' for an unbroken range of pad ID's, and 'L' for an explicit, ordered list. If most pads have data, the 'R' option is efficient (basically, marking pads without, instead of with, data); If data are sparse, the 'L' option is more efficient. Two efficiency considerations are:

1. The list ('L') option makes random pad access more expensive, since the pad position is not fixed. With 126 pads in an average row, a linear search is inefficient, but a few successive interpolations within the ordered list will suffice.
2. The range ('R') option for sparse data "wastes" 8 bytes for each unused pad.

CPU-memory tradeoffs are hard to quantify, but it seems reasonable to approximate subsectors or RDO's with ~70–90% pad occupancy as "grey," using the 'R' option.

The table entry "Pad table entries before 2^{16} offset added to 16-bit index offsets" requires some explanation. The maximum possible number of pixels having data in a pad row is 92,672, when it exceeds 65,536 for a pad, the 16-bit offset to the pixel-table index is given mod $2^16 = 65,536$, and this table entry gives the transition point, past which $x \rightarrow x + 2^{16}$.

Column	Values	Type	Bytes
Index, pixel-data tables	0 – ~2,018,000	u_long	4
Index, sequence tables	0 – ~1,005,000	u_long	4
Pixel-data entries	0 – 93,184	u_long	4
Sequence-table entries	0 – 46,592	u_short	2
Index, pad tables	0 – ~3941	u_short	2
First pad in range	1 – 182	u_char	1
Pad-table entries	1 – 182	u_char	1
Entries before 2^{16} added to index offsets	1 – 182	u_char	1
Pad referencing	N, R, L	char	1
Unused		u_char	3
Pad row ID	45 – 1	u_char	1
Total bytes per row			24

Table 3: Description of one row of the **raw_row_in**, **_out** tables, representing mapped data from one pad row in a sector. Inner subsector tables are 312 bytes, outer subsector tables are 768 bytes. Rows are inverse ordered by pad row ID: 45–14 and 13–1.

Pad Description

In the **raw_pad_*** tables described in Table 4, pads are grouped by pad row, and ordered within each group by pad number. Strictly speaking, most of the table information is redundant, but useful enough to justify the cost. Also, the desire for simple structures, combined with restrictions on the column-width mix within a row, directly conflicts with the desire to minimize storage overhead.

Column	Values	Type	Bytes
Index offset, pixel-data tables (mod 2^{16})	0 – 65,535	u_short	2
Index offset, sequence-level tables	0 – 46,155	u_short	2
Sequences on pad: N	0 – 255	u_char	1
Sequences before 2^8 added to time bins	0 – 255	u_char	1
Reserved		u_char	1
Pad ID	1 – 182	u_char	1
Total bytes per row			8

Table 4: Description of one row of the **raw_pad_in**, **_out** tables, representing mapped data from one pad. Rows are grouped by pad row number and ordered within a group by pad ID (1–182).

Having pad-level offsets to the sequence- and pixel-table row indices in **raw_row_*** allows random access to the pixel data for a pad, eliminating the need to work through the time-bin sequences from the start of the pad row. The pad ID entry is convenient, even when redundant, and should always be filled.

The number of time-bin sequences on a pad N is required if **raw_sec_m** declares that time-bin sequences are listed explicitly for the subsector. $N \leq 255$, since N is stored as a u_char variable, excluding the peculiar, but logically possible, case of 256, one-time-bin "sequences." $N = 0$ marks pads without pixel data.

Ideally, time bins would be stored as 0–511 in the sequence-level tables; they are instead stored mod 256 using 8 bits, creating ambiguity. The table entry "Sequences on pad before adding 2^8 to time bins" is the number of sequences on a pad, beyond which the first time bin $m \rightarrow m + 2^8$.

Pixel Data Description

When pixel entries fill a fixed range (black data), the above tables suffice to randomly access any pixel. In the simplest case—a standard, fixed time-bin range—all pixel data for a pad could go into one row of a pad-level table; Zero-suppressed data requires more information to specify the time bins with data.

Representing Arbitrary Time-Bin Sequences Time-bin sequences can be defined by giving the number of sequences on a pad, together with the first time bin and length of each. This is the most convenient form for cluster finding. The range of possible time bins is given in **raw_sec_m**.

Some convenient variables are:

N : Number of sequences on the pad; 0–255.
m_k : First time bin; 0–511.
m'_k : m_k, mod 256; 0–255.
i_k : Number of time bins in sequence *after* the first; 0–255.

N is available from the **raw_pad_*** table (Table 4). A **raw_seq_*** table—see Table 5—is required whenever **raw_sec_m** declares that sequences are given by list. Each of the N ordered rows for a pad gives m'_k and i_k for one sequence.

Column 1	Column 2	Type	Bytes
m'_1	i_1	$2 \times$ u_char	2
m'_2	i_2	$2 \times$ u_char	2
.	.	.	.
.	.	.	.
.	.	.	.
m'_N	i_N	$2 \times$ u_char	2
Total bytes per pad			$2N$

Table 5: Section of a **raw_seq_in**, **_out** table listing sequences on one pad. Each row gives the length (minus one) and starting position of one sequence.

Example Pseudocode for Data Access

An explicit pseudocode example is now given for accessing pixel-level data. Assume that the subdirectory for the relevant sector is selected and one wishes to access pixel-level data for the sequences on pad row r, pad p, corresponding to table rows rt, pt, respectively. An obvious scheme to extract ADC vs. time data follows. Table columns are given arbitrary, mneumonic names. The `tt_off` and `t_off` variables handle the mod 65,536 and 256 offsets, respectively.

```
/*
 * Example pseudocode to access pixel data (R. Bossingham)
 * Initialize sequence and pixel data row pointers:
 */
   st = raw_row_out[rt].IndexSequence + raw_pad_out[pt].OffsetSequence;
   rt<raw_row_out[rt].IndexPixelModBreak ? tt_off=0 : tt_off=0x10000;
   tt = raw_row_out[rt].IndexPixel + raw_pad_out[pt].OffsetPixel + tt_off;

   for (s=0, t_off=1; s<raw_pad_out[pt].N; s++){          /* Loop over sequences */
      if (s==raw_pad_out[pt].TimeModBreak) t_off+=0x100;
      t = raw_seq_out[st].m + t_off;
      i_seq = raw_seq_out[st].i;

      for (i=0; i<=i_seq; i++){                            /* Loop over time bins */
         adc = raw_pixel_adc[tt++];
         time = t++;
         /* Now do something with the ADC and time values... */
      }
      st++;
   }
```

Unmapped Data Format

Overview

The unmapped data structures, intended for electronics studies and debugging, are summarized in Fig. 4. The usual DAQ and TPC analysis data formats cannot represent "blind" channels, so these structures are mostly useful in dedicated runs producing unprocessed data that include such "blind" channels. The FEE-channel data trivially map to IC chips or 16-channel groups, as needed, and can also map onto the TPC padplane. The semi-hierarchical table relationships for unmapped data are shown schematically in Fig. 5, which may be helpful in understanding the discussion of the tables that follows.

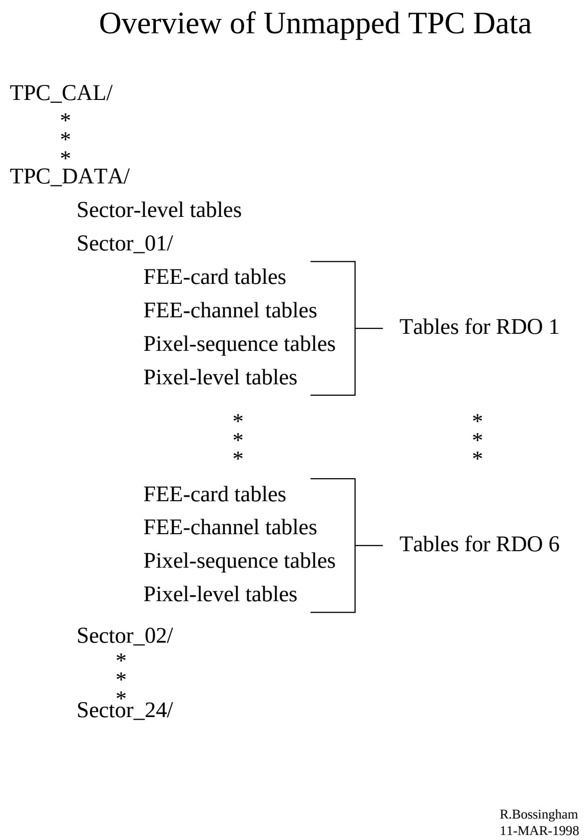

Figure 4: Overall unmapped-data structure, showing the TPC data and calibration branches, which have similar substructure.

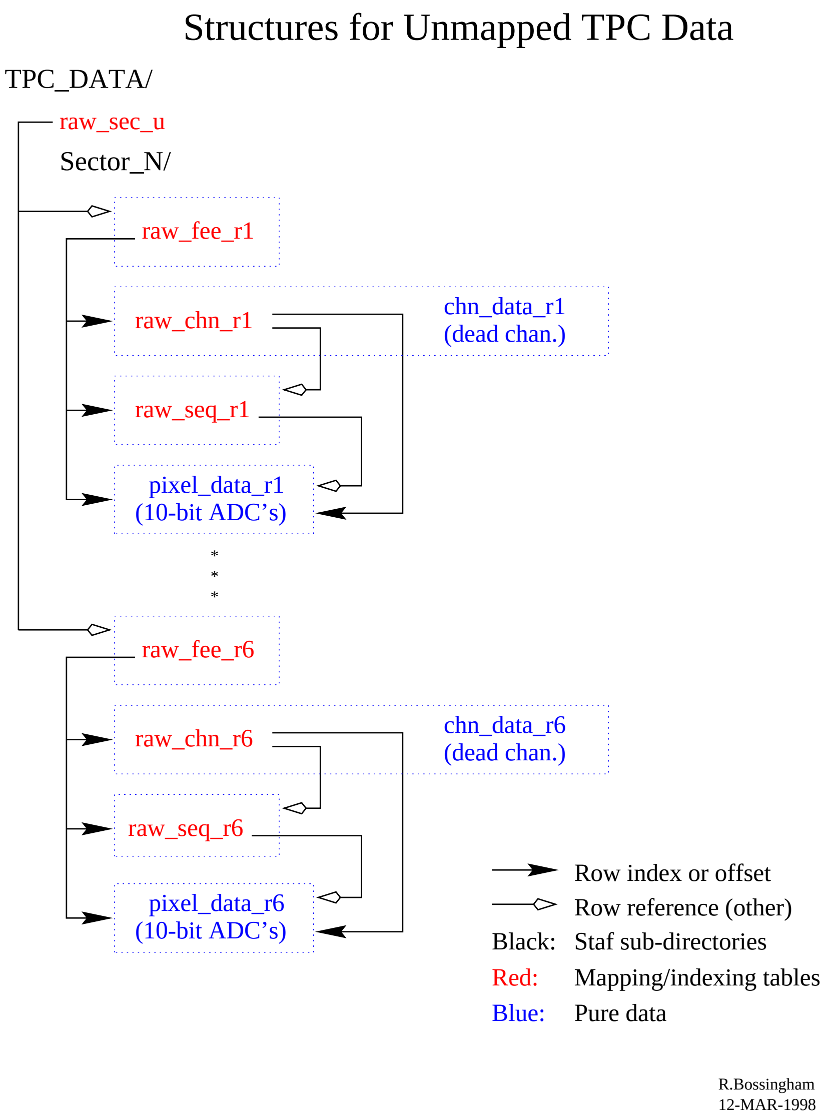

Figure 5: Structure for unmapped TPC data. The tables are discussed in the text.

Sector Description

The top-level **raw_sec_u** table—see Table 6, has exactly 24 rows ordered by sector ID (1–24). Each row gives time-bin information and describes the FEE-card tables by RDO. As for the mapped format, it is assumed that time-bin ranges and zero suppression are set at the sector level or above, and that the method of associating channel ID's with data can change below that level.

Column	Values	Type	Entity	Bytes
First time bin	1 – 512	u_short	Sector	2
Last time bin	1 – 512	u_short	"	2
Time-bin referencing	N, R, S	char	"	1
FEE-card referencing	N, R	char	RDO 1	1
"	"	"	RDO 2	1
"	"	"	RDO 3	1
"	"	"	RDO 4	1
"	"	"	RDO 5	1
"	"	"	RDO 6	1
Sector ID	1 – 24	u_char	Sector	1
Total bytes per row				12

Table 6: Description of one of the 24, 12-byte rows in the 288-byte **raw_sec_u** table. Rows are ordered by sector ID from 1–24 and define existence and interpretation of subsequent tables.

FEE Card Description

FEE-card data are described by the **raw_fee_*** tables—see Table 7—giving absolute row indices into the FEE channel, pixel-sequence and pixel tables. Giving numbers of entries in the lower-level tables is redundant, but simplifies data extraction. These tables are analogous to the **raw_row_*** tables used for mapped data, and the same options exist for referencing channels in subsequent data tables. As before, the 'R' option should be used for dense data; 'L' is better for sparse data.

FEE Channel Description

Blind channels appear in the DAQ data stream only for dedicated, black-data runs. However, zero-suppression may occur during analysis, so structures are defined here to support it. The **raw_chn_r1–_r6** tables defined in Table 8 are analogous to the **raw_pad_*** tables (Table 4), and enable access to FEE-channel data. Rows are grouped by FEE card, then ordered by channel number within a group. Below this level, sequence descriptions are identical in form to those for mapped data, using **raw_seq_r1–_r6** tables exactly analogous to the **raw_seq_in**, **_out** tables.

Columns	Values	Type	Bytes
Index, pixel-data tables	0 – ~589,311	u_long	4
Index, sequence-level tables	0 – ~293,504	u_long	4
Pixel table entries	0 – 16,384	u_short	2
Sequence table entries	0 – 8192	u_short	2
Index, channel-data tables	0 – ~4607	u_short	2
First channel in range	1 – 32	u_char	1
Channel table entries	0 – 32	u_char	1
Channel referencing	N, R, L	char	1
Unused		u_char	1
FEE card's location on sector	0 – 181	u_char	1
FEE card's location on RDO	0 – 36	u_char	1
Total bytes per row			20

Table 7: Description of one row of the **raw_fee_r1–_r6** tables, analogous to the **raw_row_*** tables, representing unmapped data from one FEE card. The tables are 720 bytes long, with rows ordered by the readout-board locations of the FEE cards (1–36).

Column	Values	Type	Bytes
Index offset, pixel-data tables	0 – 15,872	u_short	2
Index offset, sequence-level tables	0 – 7905	u_short	2
Sequences on channel: N	0 – 255	u_char	1
Sequences before 2^8 added to time bins	0 – 255	u_char	1
Reserved		u_char	1
Channel number	1 – 32	u_char	1
Total bytes per row			8

Table 8: Description of one row of the **raw_chn_r1–_r6** tables, representing unmapped data from one FEE channel. Rows are grouped by FEE card, then ordered within a group by channel (1–32).

Storage Overhead

The analysis format's storage overhead consumes processor memory and I/O bandwidth, and, temporarily (?), impacts the simulator output volume to be stored. Fractional overhead is estimated here with respect to 8-bit ADC data, but it obviously varies with data type. We use some assumptions from previous overhead discussions, and also assume:

- Negligible Staf-level table overhead.
- Zero-suppressed, mapped, 8-bit ADC data.

Central Au-Au Collision ("Standard" Assumptions)

As in previous discussions of overhead for central Au-Au events, we assume:

- $n_{\rm seq} = 4$ sequences/pad
- $L_{\rm seq} = 8$ time bins/sequence
- $N_{\rm pads} = 5690$ pads/sector
- $N_{\rm seq} = n_{\rm seq} \times N_{\rm pads} = 4 \times 5690 = 22,760$ sequences/sector

With 1080 pad rows and 136,560 pads in the TPC, the 8-bit ADC data fill

$$24 N_{\rm seq} L_{\rm seq} = 24 \cdot 22,760 \cdot 8 = 4,369,920 \text{ bytes} .$$

The storage overhead, summarized in Table 9, consists of:

- Fixed-length **raw_sec_m** table: 192 bytes
- 24 **raw_row_*** fixed-length table pairs: $24 \times (312 + 768)$ bytes $= 25,920$ bytes
- 24 **raw_pad_*** table pairs for 5690 pads: $24 \times 5690 \times 8$ bytes $= 1,092,480$ bytes
- 24 **raw_seq_*** table pairs for 22,760 sequences: $24 \times 22,760 \times 2$ bytes $= 1,092,480$ bytes

	Overhead	
	KB	%
raw_sec_m	0.2	0.0
raw_row_*	25.3	0.6
raw_pad_*	1066.9	25.0
raw_seq_*	1066.9	25.0
Total	2159.3	50.6

Table 9: Overhead for central Au-Au collisions, "standard" assumptions.

Central Au-Au Collision (Venus + tss Simulation Assumptions)

Simulations of central Au-Au events using **Venus** and **tss**[4] yield a much higher average hit density than the "standard" assumptions:

- $n_{\text{seq}} = 12$ sequences/pad
- $L_{\text{seq}} = 10.5$ time bins/sequence
- $N_{\text{seq}} = n_{\text{seq}} \times N_{\text{pads}} = 12 \times 5690 = 68,280$ sequences/sector

As summarized in Table 10, this increases ADC volume and sequence-level overhead:

- 8-bit ADC data volume: $24 N_{\text{seq}} L_{\text{seq}} = 24 \cdot 68,280 \cdot 10.5 = 17,206,560$ bytes
- 24 **raw_seq_*** table pairs for 68,280 sequences/sector: $24 \times 68,280 \times 2$ bytes $= 3,277,440$ bytes

	Overhead	
	KB	%
raw_sec_m	0.2	0.0
raw_row_*	25.3	0.2
raw_pad_*	1066.9	6.3
raw_seq_*	3200.6	19.0
Total	4293.0	25.5

Table 10: Overhead for central Au-Au collisions, Venus+tss model assumptions.

Two-Track Event with Noise

Consider a sparse event—a cosmic ray or two-meson decay in a peripheral collision—having two tracks, 45 minimum-ionizing hits each, in separate sectors. Futher assume that data from the real hits appear on three pads, and that three-time-bin noise sequences appear on 5% of all pads.

- $n_{\text{seq}} = 1$ sequence/pad
- $L_{\text{seq,real}} = 5$ time bins/sequence for real hits
- $N_{\text{pads,noise}} = 0.05 \cdot 5690 = 285$ pads/sector (24 sectors)
- $N_{\text{pads,real}} = 45 \cdot 3 = 135$ pads/sector (2 sectors)

The 8-bit ADC data consume

$$\sum_i L_{\text{seq}_i} N_{\text{pads}_i} = 2 \cdot 5 \cdot 135 + 24 \cdot 3 \cdot 285 = 21,870 \text{ bytes} ,$$

[4] Iwona Sakrejda, http://www.rhic.bnl.gov/star/starlib/doc/www/html/tpc_l/tpc.html (1998).

and the storage overhead, summarized in Table 11, consists of:

- Fixed-length **raw_sec_m** table: 192 bytes
- 24 **raw_row_*** fixed-length table pairs: $24 \times (312 + 768)$ bytes = 25,920 bytes
- 24 **raw_pad_*** table pairs for 7110 pads (total): 7110×8 bytes = 56,880 bytes
- 24 **raw_seq_*** table pairs for 7110 sequences (total): 7110×2 bytes = 14,220 bytes

Hierarchical structures are intrinsically inefficient for sparse events, especially here, where the granularity and pointer lengths were chosen to support dense data: a simple list describing all sequences in the TPC would have less overhead. The exact numbers are sensitive to threshold settings and other assumptions, but 95 KB (~440%) is a plausible guestimate.

Table	Overhead	
	KB	%
raw_sec_m	0.2	1
raw_row_*	25.3	119
raw_pad_*	55.5	260
raw_seq_*	13.9	65
Total	94.9	444

Table 11: Estimated overhead for noisy, 2-track events.

Two-Track Event *without* Noise

Since the Level-3 CPU's might be used to strip noise hits from the data,[5] consider the same sparse event without noise. The 8-bit ADC data now fills

$$\sum_i L_{\text{seq}_i} N_{\text{pads}_i} = 2 \cdot 5 \cdot 135 = 1350 \text{ bytes} .$$

The storage overhead, summarized in Table 12, is strongly reduced by the noise removal:

- 2 **raw_row_*** fixed-length table pairs: $2 \times (312 + 768)$ bytes = 2160 bytes
- 2 **raw_pad_*** table pairs for 270 pads (total): 270×8 bytes = 2160 bytes
- 2 **raw_seq_*** table pairs for 270 sequences (total): 270×2 bytes = 540 bytes

Black Event Overhead

The fractional overhead for black (e.g., pedestal) data is negligible (< 0.1%), due to the enormous data volume and its consistent structure; any "reasonable" data format will do. In any case, relatively few such events will be stored, so the low overhead of the proposed format for black events is a minor consideration.

[5]Spencer Klein, private communication.

	Overhead	
	KB	%
raw_sec_m	0.2	14
raw_row_*	2.1	160
raw_pad_*	2.1	160
raw_seq_*	0.5	40
Total	5.0	374

Table 12: Estimated overhead for 2-track events without noise.

Comparison of Overhead for Various Formats

For dense data, the hierarchical structures defined in this Note incur much less overhead than the 325% required by the tppad/tppixel structures, translating to a factor of three reduction in the size of central Au-Au events. Distributed data volume and overhead (central Au-Au, Venus+tss) are compared in Table 13 for various raw data structures. Overhead, alone, and total volume (overhead+data) are both interesting, since operations like cluster finding can operate purely on the overhead tables, while operations like event distribution must handle the full volume.

	Total	Overhead	
	KB	KB	%
tppad/tppixel	71,500	54,700	325
SN0325, V1.0	20,800	4,000	24
Iteration 1	24,300	7,500	45
SN0325, V2.0	21,100	4,300	26

Table 13: Overhead for central Au-Au collisions, under Venus+tss model assumptions, for several studied formats. The tppad/tppixel tables merge overhead and 10-bit ADC data; so numbers here have been compensated for this. Compared to this Note, "SN0325, V1.0" reduced overhead and increased complexity by using variable-length and optional tables. "Iteration 1" reduced complexity by using wider variables to eliminate the breakpoints—at the cost of considerably higher overhead.

Tables to Store Clusters from the i960's

Clusters are found by the i960's from the raw data, so it is not restrictive to require that clusters appear only in pad rows with raw-data entries when the event includes raw data. The i960 clusters can then be put into the raw data directories, but still need separate, lower-level structures, as in Table 14.

The i960 clusters within a pad row are described by the **c96_row_*** tables defined in Table 15. Absolute indices into the cluster tables are given. As before, giving the number of lower-level table rows corresponding to a pad row is redundant, but simplifies the extraction of data subsets.

The i960 cluster description should resemble the DAQ format as closely as possible; the **c96_cls_***

Hierarchy	Mapped Data Table
Full TPC	**raw_sec_m**
Sector_N/ Level 1	**c96_row_in, out**
Sector_N/ Level 2	**c96_cls_in, out**

Table 14: Tables to represent and store the i960 clusters.

Column	Values	Type	Bytes
Index, i960 cluster tables	0 – ~190,000	u_long	4
i960 cluster entries for pad row	0 – 6000	u_short	2
Reserved		u_char	1
Pad row ID	45 – 1	u_char	1
Total bytes per row			8

Table 15: Description of one row of the **c96_row_in**, **_out** tables, representing clusters found by the i960's in one pad row of a subsector. Rows are inverse ordered by pad row ID: 13–1 or 45–14.

Column	Type	Bytes
Centroid in X (1/16 pad units)	short?	2
Centroid in T (1/16 time bin units)	short?	2
Charge total	u_short?	2
Width in X (1/4 pad units)	u_char?	1
Width in T (1/4 time bin units)	u_char?	1
Total bytes per cluster		8

Table 16: Description of one row of the **c96_row_in**, **_out** tables, representing an i960 cluster. Clusters are grouped by pad row, but not (necessarily) otherwise sorted.

tables—see Table 16—use the obvious form (although a preliminary DAQ format write-up seemed ambiguous as to whether some of the variables were signed or unsigned.) For a central Au-Au collision, using "standard" assumptions and three sequences per cluster, we find the following increase in data volume to store the i960 clusters:

- 24 **c96_row_*** table pairs: $24 \times (13 + 32) \times 8$ bytes = 8640 bytes
- 24 **c96_cls_*** table pairs with 7587 clusters per pair: $24 \times 7587 \times 8$ bytes $= 1,456,704$ bytes

Comparing this 1431-KB requirement with the 4268-KB volume of the 8-bit ADC's and the 3222-KB overhead for the same "standard" assumptions, the overall increase is 19%: significant, but not overwhelming. Nonetheless, one should specify to the event server whether raw data, i960 clusters, or both, are needed in an event. In general, there will be several types of processed data from the i960 or Level-3 processors, and a general framework should be defined to request types of data.

Overhead in STAR tppad/tppixel Format

The tppad/tppixel format, formerly used for TPC Slow Simulator (**tss**) output and still used in the TPC CLuster finder (**tcl**), is storage- and CPU-inefficient: it contains considerable redundant information and empty space, and requires bit-level decoding. An earlier estimate[6] of its storage overhead is reproduced here.

Each pad is indexed with four long int's, and each pixel uses an additional long int:

```
struct tss_tppad{
    long      jpix;       /* offset into the tppixel table */
    long      nseq;       /* number of sequences for this pad */
    long      secpad;     /* unique pad id within a sector */
    long      tpc_row;    /* encoded sector-row = 100*sector + row */
};

struct tss_tppixel{
    long      datum;      /* encoded tzero, nbins, adc values */
};
```

Total storage for a central Au-Au event ("standard" assumptions) is

$$(4 \cdot 4 + 4N_{\mathrm{seq}}L_{\mathrm{seq}})N_{\mathrm{pads}} = (16 + 4 \cdot 4 \cdot 8) \cdot 136,560 = 19,664,640 \text{ bytes} .$$

Since the format allocates 10 bits for each ADC, these occupy 5,462,400 bytes, leaving 14,202,240 bytes of overhead—10.2× as much as the format defined in this Note.

[6] SN0282, R.Bossingham (1997.

Application Considerations

Despite the preceding discussions of CPU and storage overhead, the most important questions about data structures concern their use in real applications: how do they affect their design and coding? To *really* answer this requires actual coding and testing, but we can do a few, simple *gedanken* experiments here.

Event Server Considerations

An on-line event server will provide subsets of the TPC data to consumers, so the designs of the data structures and event server are inter-related, involving several issues:

1. Granularity supported by the event server. This should not be below the pad level, or above the sector level. At the pad level, the data request would be dominated by the cost of extraction; at the sector level, there would usually be a significant cost in I/O bandwidth. Information on a single pad is of limited value, but one can easily imagine wanting data below the sector level—a few outer pad rows for multiplicity or distortion estimates, for example.

   Suggestion: Support data requests at the granularity of a pad row or FEE card.

2. Form of events stored in the server. If events are stored in DAQ format and translated separately for each data request, it is always cheaper to translate and send a smaller data unit. This scheme would be justified only if most requests are for small subsets of an event. On the other hand, if DAQ data are translated and stored in the analysis format, the cost of extracting data must be balanced against the cost of transmitting unneeded data.

   Suggestion: Store translated events to reduce latency, and define the event format to minimize the cost of extracting data at the supported granularity.

3. Event re-access through the event server. Events could be cached on the event server, allowing a user to access additional data, until a new event is requested. This is practical if (and probably only if), the number of simultaneous data consumers is less that the number of events that can be stored on the server at one time. The system would allow a user to be conservative about the volume of data requested interactively: less I/O, at the expense of more event storage.

   Suggestion: Desireable, if practical...

4. Simultaneous support for mapped and native data. DAQ will only output data from blind FEE channels in special runs dedicated to electronics study and diagnostics. This data can be provided to a consumer only in the native format, raising the question of whether the event server should simultaneously support native and mapped events or, alternatively, who translates into the mapped format?

Suggestion: When producing native-format data, simultaneously produce and store the mapped-format data.

5. Degree of complexity supported for event server requests. Can any arbitrary combination of data down to the supported granularity be requested?

   Suggestion: Support all possible combinations of legal subsets.

6. Form of data requests to the event server. The appropriate choice depends upon other design choices—supported granularity, complexity of requests, etc.; requests for data only at the sector level can be supported trivially, but describing an arbitrary data subset requires a complexity approaching that of a data format.

   Suggestion: An interesting idea would be to make the request in the form of the data structure to be filled.

On-line Extraction of Data Subsets

For the sake of discussion, assume that the on-line event server incorporates the above suggestions. Also assume that, at some given time, DAQ is providing mapped, zero-suppressed data from all sectors; the event server is producing mapped "grey" data (data on all pads); and that an appropriate Staf directory structure exists (maybe passed to the server as the data request). Consider the process of extracting some data subsets:

Sector N, outer subsector: Easy; most tables are subsector tables. Update **raw_sec_m**; then copy all of the **TPC_DATA/Sector_N/*_out** tables, and any tables for RDO's 3–6.

Sector N, pad rows 44, 43: A bit harder; update **raw_sec_m** and go to **TPC_DATA/Sector_N**. Copy the appropriate parts of **raw_pad_out**, **raw_seq_out** and the pixel data for pad rows 44 and 43. No absolute indices appear in these tables, so no rows are modified. When **raw_row_out** is copied, the indices and offset breakpoint are adjusted for pad rows 44 and 43, and the number of pad entries is set to zero for the other rows. We probably still copy any tables for RDO's 3–6.

Sector N, pad row 44, pads 70, 71, 72: Still harder; pad-level access is probably as microscopic as one should expect from an event server, philosophically, but this is still tractable because of the pad-level indexing. The index offsets in **raw_pad_out** have to be recalculated, as do all of the entries for pad row 44 in **raw_row_out**.

Sector N, RDO 6: This request should probably return an error code, since we assumed that the event server is producing mapped data, making data from blind channels unavailable. One *could* return the RDO 6 tables (e.g., header tables) or even remap the available data.

Data Pre-processing Considerations

In general, the data must be pre-processed before cluster and hit finding: 8-to-10-bit translation; pedestal subtraction; gain correction; threshold application; zero suppression; and/or embedding Monte Carlo tracks into real data. We examine these problems briefly to determine whether or not our structures are compatible with practical solutions.

One-to-One Table Mapping Operations

Suppose that a simple function exists that maps each entry in a table into an entry in a parallel table; an obvious example is 8-to-10-bit translation. (Staf doesn't support 10-bit integers, so the actual translation would be from u_char's to u_short's.) One would get the row count for the 8-bit table from Staf and allocate a table for the u_short's, then loop over the rows, translating into the new table. At the end, the old, 8-bit table could be either kept or deleted. No operations on the overhead tables are necessary.

Similarly, a fixed function may be applied to the entries within a table, such as the application of a fixed threshold to "black" data. In this case, one loops over the rows, zeroing sub-threshold values. Again, no higher level tables have to be accessed.

If the mapping function has dependencies, as a gain correction or pedestal subtraction would, one must use the higher level tables to associate a particular pad or pixel with each table entry. Simultaneously, one would probably have to work through another, similar set of tables to get the applicable gain or pedestal. In principle, the necessary information could be put into tables parallel to the ones being modified, but this would usually just shift the same work to a different place.

More Complex Table Mapping Operations

More work is required if the mapping function obsoletes the structure. For example, threshold application may insert empty time buckets into sequences, shortening or splitting them. In general, this requires a full recalculation of the overhead tables. An example would be for sequences to be analyzed by a fast cluster finder based on topology alone, in which pixel contents were not examined.

This particular problem doesn't appear for the present **tcl** cluster finder; it uses the tppad/tppixel structures only for storage, copying the data to working arrays before any operations, so any reasonable format would serve as well. In particular, **tcl** copies sequences into a flat field of zeroes before pattern-finding, then copies the pixels from each cluster into a new structure. On the other hand, the cost of constructing the new structures and of the extra data handling is undoubtedly more than the cost of reconfiguring the old structure.

Conclusions

The structures defined in this note can be summarized as highly hierarchical, with the possibility of infinite parallelism at the level of the data tables. This allows both storage efficiency and flexibility. In general, the structures are compatible with STAR's needs for event serving and cluster finding, and should facilitate their implementation/improvement. For dense data, the structures offer significantly lower overhead than the 325% required by the tppad/tppixel structures now in use, translating to a factor of three reduction in data volume for central Au-Au data.

It is difficult to quantify the tradeoff between programming ease and machine efficiency, but I believe that the format passes a "reasonable programmer" test—i.e., defining a simpler or more efficient format would be unlikely to reduce the overall programming task or CPU and storage requirements significantly.

Not all necessary conventions have been defined in this Note; in some cases, the format allows more than one implementation. These decisions are left until various design factors are better defined.