The CDF Run II Event Data Model

presented by Robert D. Kennedy
for the CDF Event Data Model Working Group
CHEP 2000 (Padova, Italy), short talk C201

Event Data Model: Manages reading/writing of events to/from data files, and manages access to objects in an event.

Data Handling System: Manages file repository, delivers files.

Framework: Directs execution/configuration of software modules.

Why a New Event Data Model? Run I EDM was successful, but....

Introduction to Run II EDM: EventRecord, StorableObjects

ROOT Object I/O: How do we use it for analysis, on farms?

Status and Future Work: Core new EDM done, Off-line adapted.

<u>time permitting</u>
More Components: Links, StorableContainers, StorableBanks
Fortran-77 Support: Banks transformed to/from StorableBanks
Sample Use-Cases: The look and feel of code in new EDM

---

http://www-cdf.fnal.gov/upgrades/computing/projects/edm/edm.html

Why a New Event Data Model?

CDF Run I EDM: Data were stored in YBOS/Trybos banks. Banks labelled by 4 character name and by a number assigned by user. Event record was a global array storing YBOS banks, YBOS internals, and (for F77 code) temporary work space.

Common Usage: User gets an array index for first bank with name == "MUON", and uses index plus offsets (defined in F77 include file) to access fields in MUON bank. Then, the user gets the array index to the next bank with same name.

Problems: Corruption of event array when writing with wrong offsets, change of an associated object without updating the dependent objects, bank numbers re-used (object's history?), discomfort using banks led to widespread use of common blocks.

Contemporary Trends: More and more of CDF Off-line code is written in Object-Oriented C++ using a C++ re-implementation of YBOS, but which still only supports YBOS-format banks.

Decision to Use ROOT Object I/O: After an exhaustive review, ROOT Object I/O was selected to implement event data storage.

The Run I Event Data Model was a success, but...:
We thought we could do better, support more general C++ objects, and still not discard existing code and data files.

Run II EDM Goals and Constraints

- ☞ Retain as much existing C++ code as possible.
- ☞ Retain ability to read existing data files.
- ☞ Improve reproducibility, manageability, ...
- ☞ Support more general C++ objects in event.
- ☞ Use ROOT Object I/O for persistence.

EventRecord and StorableObjects (1)

☞ **All data passed from module to module is passed via an instance of the class *EventRecord*.**

No singletons or globals may be used to pass event data.

Not enforced in Run I. Some exceptions in Simulation software.

☞ **Objects derived from class *StorableObject* can be stored in the event record.**

The event is an STL-based container of StorableObject pointers.

☞ **Storable objects must be allocated on the heap and referred to using instances of *Handle* classes.**

A Handle is a smart pointer class, used to avoid data copies.

☞ **Storable object classes must implement a few methods, such as Streamer(), which perform serialization and such tasks involved in event I/O.**

☞ **Storable object classes must be parsable by rootcint, and yield a valid I/O dictionary entry.**

StorableObject is derived from ROOT's *TObject*, but adds some CDF-specific protocol. The C++ header and a simple "linkdef" text file for each class are used to drive rootcint.

☞ **Objects are assigned an object id by the event when stored, and then become read-only.**

Big change from Run I which affects some existing software.

Each object is "frozen" in record, preserving its history and the validity of associations of other objects to it.

EventRecord and StorableObjects (2)

☞ **Users cannot delete objects in the event record. They can only classify the object as being desirable or undesirable for output.**

This ban on object deletion is new, but the use of I/O "keep" and "drop" lists is not. This does affect some existing algorithms.

☞ **The creating module (and its parameters) of an object is recorded in an "rcp id" in the object.**

Rcp id is used to get module meta-data from an RCP database. The RCP system is not yet integrated with the new EDM code.

☞ **Users can search for objects in the event by class name, object id, rcp id, a user-defined descriptive string field, or boolean combinations.**

Iterators over objects in event remember the specified selection criteria. STL-like style can be used for loops over objects.

☞ **A *StreamableObject* class does not satisfy the criteria of a *StorableObject* class, but does define a Streamer() method. These objects can only be stored in an event indirectly if they are contained by an instance of a valid *StorableObject* class.**

Some *StreamableObject* classes choose to be small non-virtual classes not deriving from *StorableObject*, such as an id class. In other cases, rootcint cannot parse the class properly, such as template-based homogenous container classes. In any case, some storable object must know to call their Streamer() inside its own.

Use of ROOT Object I/O in Analysis

☞ **ROOT supplies I/O facility for user-defined objects which preserves the class/type of objects.**

To read an object, ROOT first reads the class name and locates an entry containing pointers to that class's methods in ROOT's I/O dictionary. ROOT constructs an instance of the object, deserializes the object from disk using its Streamer(), and returns a pointer to the restored object which the EDM puts into event's object list.

☞ **CDF's sequential access-oriented data file class, *SeqRootDiskFile*, models all event data in one ROOT *TBranch* in one ROOT *TTree*.**

We nominally access all event data during event reconstruction, so there is limited advantage to storing events in many branches at this stage of processing. We do not yet support random access of events in a sequential data file. Also, users must use disk files for all direct I/O. The Data Handling system provides tape access.

☞ **Objects are stored as a whole. Individual data members are not stored in separate branches.**

CDF plans to exploit multi-branch data files in the future for secondary and tertiary data sets. The use patterns of different categories of physics studies will guide how classes will be assigned to the $O(10)$ branches we might use. Different physics groups will independently define and record this assignment.

☞ **Object browsing is a low priority. Some support will eventually be integrated into EDM.**

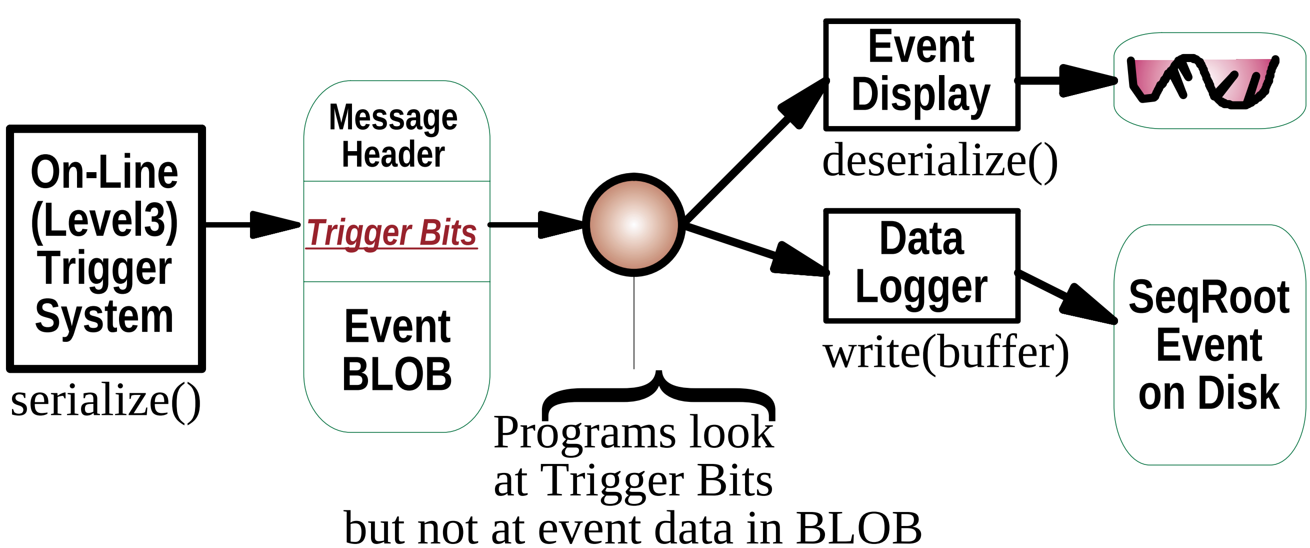
Use of ROOT Object I/O on Farms

☞ **An event may also serialize itself into a user-allocated buffer for transport as a BLOB, and can later be deserialized from that BLOB.**

On computing farms, we often ship events from one process to another without the processes needing access to the data inside the event. We wish to avoid the overhead of serializing and deserializing events repeatedly. We re-use the serialization protocol in EventRecord::Streamer() with a buffer of our own making, rather than one tied to the ROOT Object I/O system.

☞ **A serialized event is capable of reading/writing itself in the same *TTree*/*TBranch* model as a SeqRoot event, one buffer for each branch.**

This capability is waiting for an extension by the ROOT team. This would permit us, for instance, to transfer serialized raw data events over the network, and then write them to disk in the same format as events read/written in analysis without the overhead of fully deserializing the event in the receiving process.



Status and Future Work

☞ All of the core components of the CDF Run II Event Data Model have been implemented.

Though some classes are not yet fully optimized, all pieces exist to access event data in a reconstruction or analysis program.

☞ All Off-line event reconstruction code has been adapted to use the CDF Run II EDM.

Tracking consumes about *40% less* CPU in new EDM than in old... no more unpacking to common blocks. Some Simulation, On-Line code has not been adapted yet, as well as much user analysis code.

☞ CDF has completed its first Mock Data Challenge using the CDF Run II EDM.

Simulated events were read by the high-level on-line triggering system, selected, sent to reconstruction farms, reconstructed, and split into analyzable sub-samples. All code used the new EDM.

☞ But, more work to be done on the EDM.

As more users use the new EDM in analysis code, we expect to identify desirable extensions, improve make procedures, etc. Much work to support multi-branch split-file tertiary data sets....

Tasks for the near future

Optimization: reduce CPU used for I/O, improve memory mgmt
Rubustness: improve error-handling and make procedures, etc.
Adaptation: help in the adaptation of user analysis software
Documentation: complete new user and reference guides

Links, StorableContainers, StorableBanks

☞ **A *Link* class is a smart pointer class which can save/restore its state to disk. To work, a *Link* must point to a storable object.**

A *Link* is a dual pointer and object identifier. Upon being streamed out, a *Link* streams out the oid of the thing to which it points. Upon being read back in, a *Link* can restored to its original state during a post-read phase by searching for the object pointed to by its oid.

☞ ***StorableContainers* are a number of streamable object jackets on STL container classes. Access is provided to underlying STL container so that STL iterators and algorithms can be used.**

Containers exist for vectors and lists, and for storage by value, by owning reference, and by non-owning reference. A storable Track class may contain, for instance, a ValueVector<Hit>, and a reconstruction module may produce a RefVector<Track>.

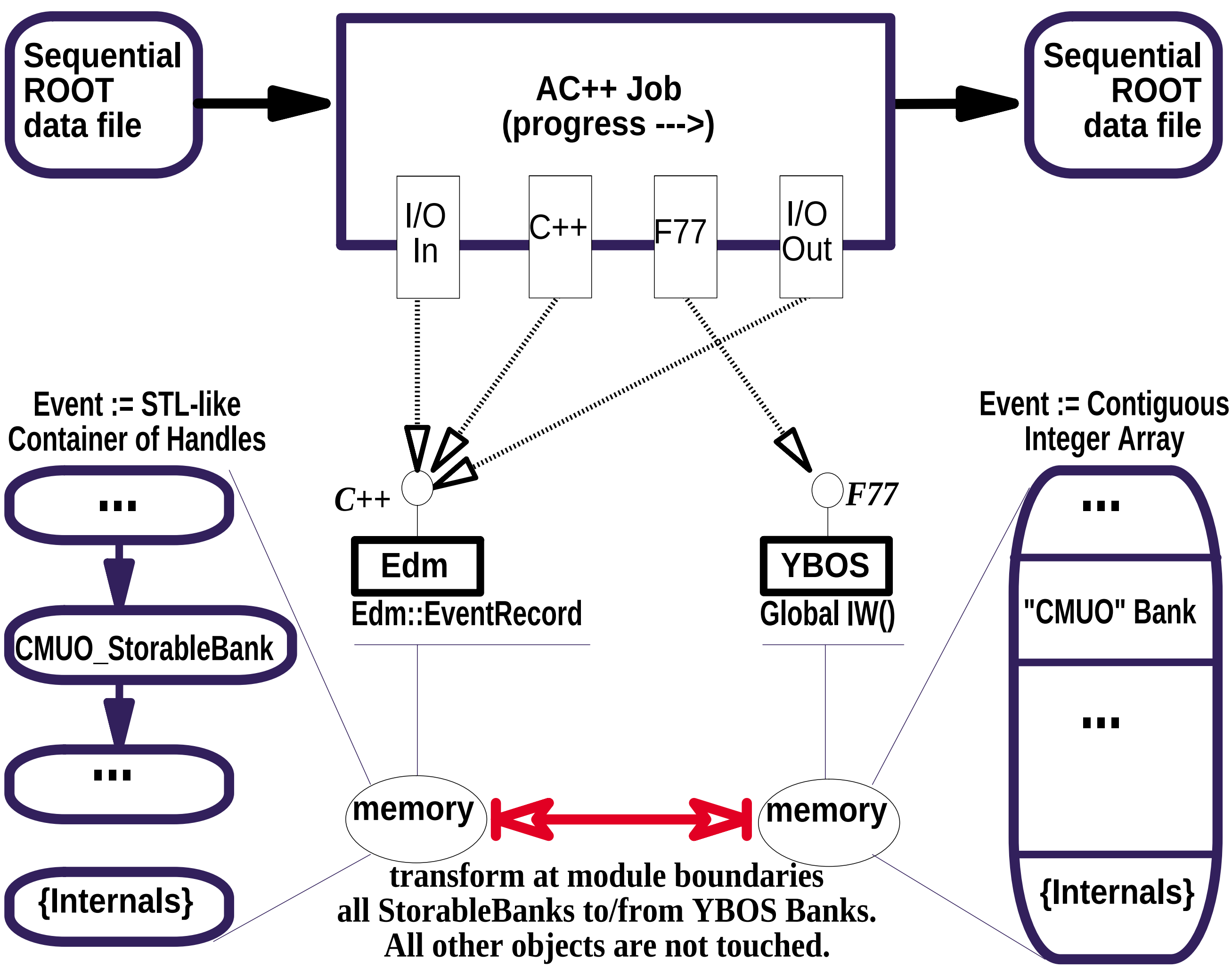
☞ ***StorableBank* is a base class for YBOS banks.**

***StorableBank* is the new EDM analog of the C++ encapsulation of a YBOS bank, *TRY_Generic_Bank*. The same basic API is supported by *StorableBank* to simplify both the adaptation of 100+ Banks classes to the new EDM and to make efficient the transformation of Generic Banks to/from StorableBanks.**
New EDM equivalents exist for data iteration support for banks with specific internal data structure, mostly raw data banks.

Fortran-77 Module Support

☞ **A Fortran-77 module will still operate if it only inputs and outputs Banks (no globals).**

The module requires some wrappering in order to cause the storable banks in a new EDM event to be transformed into and back from a YBOS global array of banks. This process weakens some aspects of the new EDM since there is no way to prevent Fortran-77 modules from modifying existing banks and so on.



Sample Use-Cases

☞ Create a *ToyTrack* and *ToyMuon* object

```
// Create a track and a muon, setup track
Handle<ToyTrack> wtrk(new ToyTrack) ;
Handle<ToyMuon>  wmuon(new ToyMuon) ;
wtrk->set_track_id(1) ;
wtrk->set_algorithm_name("Fake-by-hand") ;
wtrk->set_chisqr(10.0) ;
wtrk->set_nhits(151) ;
// Add the track to the event
GenConstHandle rtrk = p_event->append(wtrk) ;
// setup muon with link to this track
wmuon->set_em_charge(+1) ;
wmuon->set_4momentum(1.0, 2.0, 3.0, 4.0) ;
wmuon->set_intercept(0.1, 0.2, 0.3) ;
Link<ToyTrack> track_link(rtrk) ;
wmuon->set_track_link(track_link) ;
// Add muon to the event
GenConstHandle rmuon = p_event->append(wmuon) ;
```

☞ To update or revise an object in the event:

Initialize a newly allocated object with one from the event.
Modify the new object and append it to the event.
Non-owning reference lists can help avoid some excessive copies.

☞ To "delete" an object from the event:

User can add object id to "drop" list. The event output method consults this list before outputting each object in the event.