Introduction to Trybos - Part 1
Goals, Design, and Status

Rob Kennedy

Original: 03-Jan-96
Revised: 31-Jan-96
for Trybos Version 2.1 Release

Trybos (Totally Revised YBOS) is an Object-Oriented C++ package to manipulate YBOS formatted data. This talk describes some of the project goals, the overall package design, and the current design and implementation status.

Presentation to the CDF Computing Upgrade Group

Project Goals

Goal: Accessing YBOS-Format Data from C++

Provide Object-Oriented C++ interface to better mate with future C++ software development source.

Produce EQUIVALENT binary output given the same data as YBOS. The output files from Trybos and YBOS can be read by either package without source code or data modification.

Avoid clumsy, inefficient, and error-prone model of all data stored in a flat, unprotected, and global array.

Re-organize and re-design in order to improve overall efficiency.

Note: Even without going to Object-Oriented C++, these goals can be achieved. Earlier versions of Trybos in the C language, for example, showed far better modularility and localization in the source code (re-use at the design as well as implementation level).

Goal: Exploring Package-Independent Interfaces

Define a “clean” interface for YBOS, then consider how to modify or extend to accommodate alternative persistence packages.

NOT Goal: Produce Pin-out Replacement for YBOS

Trybos is not intended to be an exact replacement for YBOS in design, nor in programmer interface. In order to meet the above goals, it is in fact close to reproducing the YBOS programmer interface in principle, but not in actuality... and it is not a goal to do so.

Package Design

<u>Design Clusters: The Conceptual Partitions in Package</u>

A) **Bank** of typed data

1) <u>Bank Key</u> - unique identifier of a Bank within a Record
2) <u>Generic Bank</u> - knows about above and data type information. Most primitive class for a Bank not stored in a Record
3) <u>Const Specific Bank</u> - knows about above and data fields, BUT data fields can only be read (accessors are “get-only”).
4) <u>Specific Bank</u> - same as Const Specific Bank, except that one can set as well as get data field values

B) **Record** is a collection of Banks with unique Bank Keys

1) <u>Record Iterator</u> - Points to a Generic Bank within a Record. Knows how to advance to next Bank, when “end” is reached.
2) <u>Basic Record</u> - simple flat array is used to store Banks, optimized for I/O rate at the expense of Bank finding speed and Bank insert/erasing speed. Used for all I/O operations.
3) <u>Hashed Record</u> - simple flat array is used to store Banks, optimized for sequential Bank finding speed at the expense of I/O rate. Uses a Hash Table of first Bank locations. Similar to “usual” YBOS package logical record.
4) <u>Listed Record</u> - simple flat array plus some specialized buffers are used to store Banks. Optimized for Bank insert/erase and find operations at the expense of I/O rate. Uses Hash Table and unusual List to speed-up most Bank manipulations.

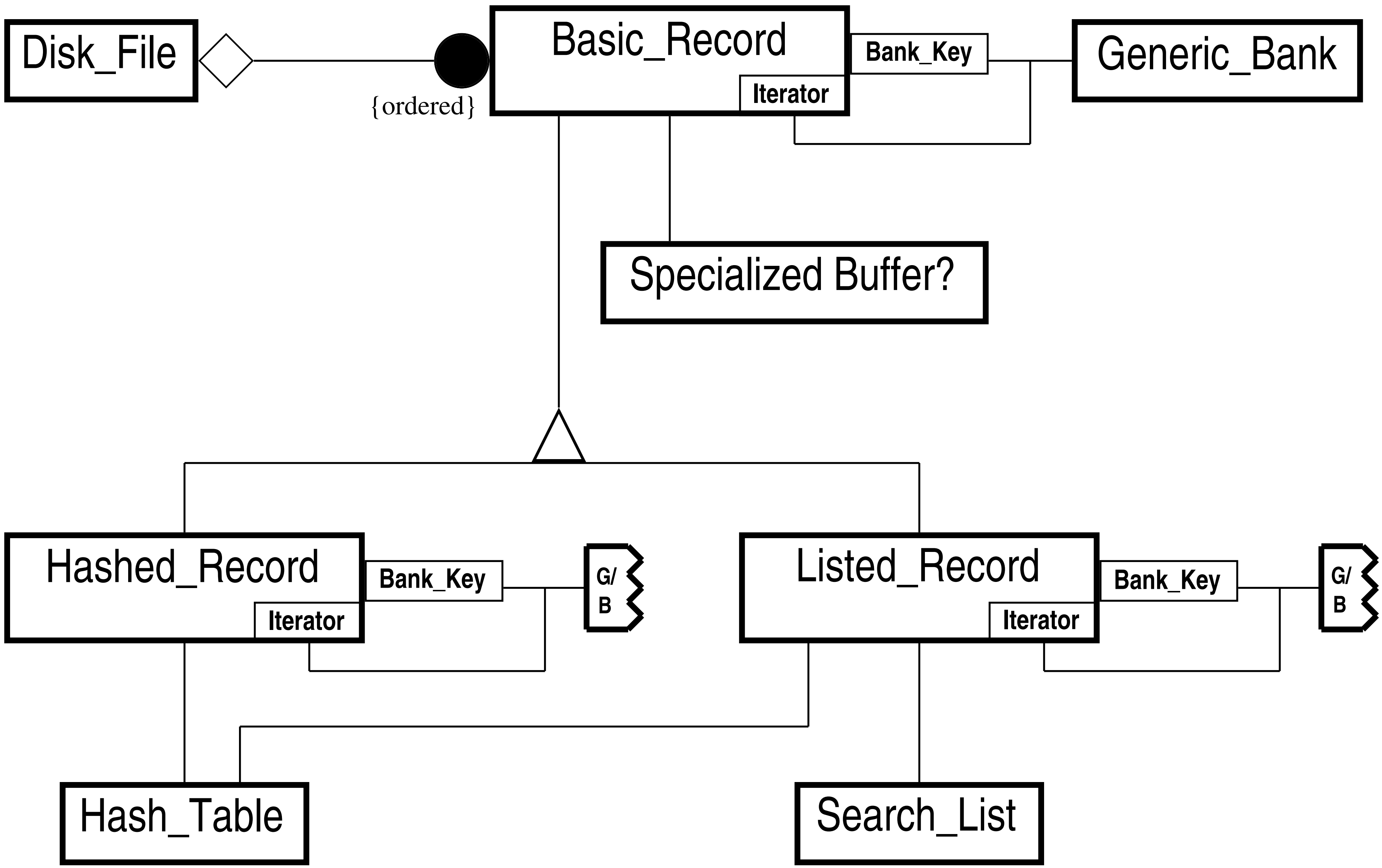
C) **File** is a collection of Records

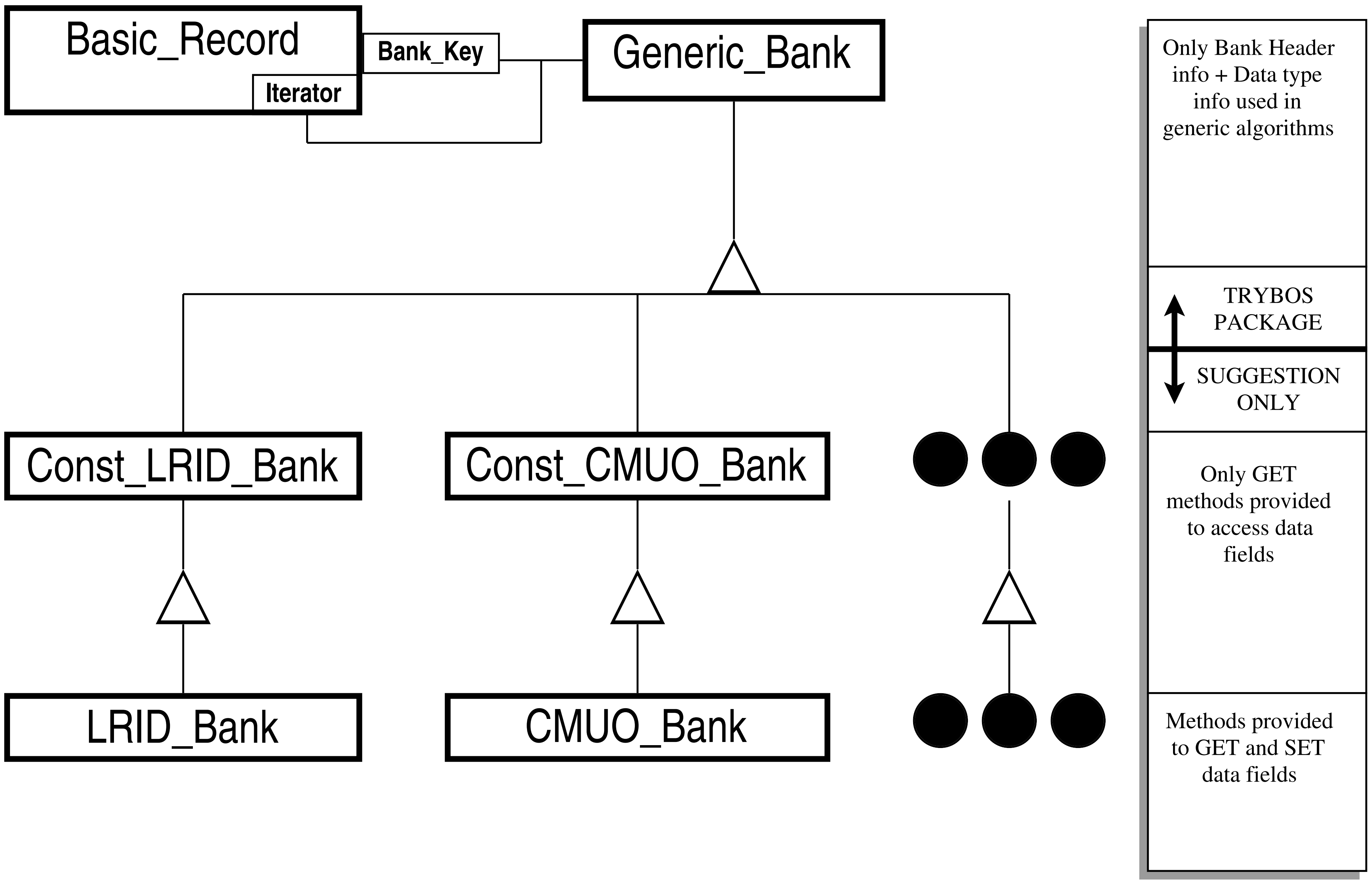
1) <u>Disk File</u> - only File class supported at present

D) **Dataset** is a collection of Files

0) No classes support this abstraction at present

OMT Class Diagram
Record Cluster and Relations




Basic_Record
Bank_Key
Iterator
Generic_Bank
Const_LRID_Bank
Const_CMUO_Bank
LRID_Bank
CMUO_Bank
Only Bank Header info + Data type info used in generic algorithms
TRYBOS PACKAGE
SUGGESTION ONLY
Only GET methods provided to access data fields
Methods provided to GET and SET data fields

Design/Implementation Issues

Design Issues

1) **Banks**

Make a choice between the following:
a) crude translation of F77 offsets .OR.
b) revise programmer interface to match actual data values .OR.
c) revise programmer interface to better design with same data

Also, separate minimal Bank description from current Specific Banks? This would improve package independence but increase the support burden for the short-term.

3) **Record** is a collection of Banks

Choice of which Record to use is left to the programmer and depends on the context (Production, Level 3, or User analysis). Perhaps this can be a grand switch somewhere?

2) **File**

Concept of Physical Record is hidden in design and implementation because of its UGLINESS. Make explicit in design, but continue to hide in implementation... make explicit everywhere?

Implementation Issues

Compiler = GNU C++ 2.7.1,2 on SGI IRIX 5.2, NeXTStep 3.3.1
GNU C++ 2.7.0 fails, cannot assign structures (bug)

Method names tend to follow C++/STL scheme and not CDF YBOS scheme. This can be changed later by providing “alternative protocols” which present some combination “basic methods” in a different guise and are inlined to maintain efficiency.

Design/Implementation Status

Design/Implementation Status

A) **Bank** partition

*) Bank Key, General Bank - DONE

*) Const Specific Bank, Specific Bank - LRID, CMUO, no JPSI

B) **Record** partition

*) Basic Record, Hashed Record - DONE

*) Listed Record - Interface is done, but the specialized Search List class only loosely designed

C) **File** partition

*) Disk File - DONE

D) **Dataset** Partition - no design work yet

Miscellaneous Status Details

*) LRID Bank - a set() method missing for creation time...

*) Basic Record, Hashed Record- erase(), insert() untested

*) Hashed Record- erase(),insert() do not update next_bank index yet

Version 2.1 Release Briefs

1) **Goal** is to get something “out there” for people to tinker with.
The interface must be agreed upon before code using Trybos is written for serious upgrade projects. The implementation can evolve... already 25-40% more efficient than YBOS.

2) **Where**: cdfsga as a UPS product... “setup trybos”

3) **Documentation** : This presentation and...
“The Trybos Document”, Cdf/Doc/Comp_Upg/0000

4) **Archive** not yet made of files... wait for first patch or lack thereof

5) **Compiler** - GNU C++ 2.7.1 or 2.7.2 (2.7.0, in particular, is buggy)

6) **What’s Next**: People critique interface, internal improvements, doc

Example of Source Code Driving Trybos Design

```
Trybos_Basic_Record record ;  Trybos_Disk_File input_file ;
integer4 jpsi_total = 0 ; // declare and init jpsi_mass, window_halfwidth
// OPEN INPUT_FILE...
while(input_file.read(record))  // Read a Record from a File into "record"
{integer4 jpsi_index = 0 ;
  Trybos_Basic_Record::Iterator p_CMUO_1 = record.find_first("CMUO") ;

  while (p_CMUO_1 != record.end())
{Trybos_Basic_Record::Iterator p_CMUO_2 = record.find_next(p_CMUO_1, "CMUO");

    while (p_CMUO_2 != record.end())
{CMUO_Const_Bank CMUO_1(p_CMUO_1) ;  // dress up Pointer as a Stored Bank
      CMUO_Const_Bank CMUO_2(p_CMUO_2) ;  // dress up Pointer as a Stored Bank

      double pair_mass = (CMUO_1.p4() + CMUO_2.p4()).magnitude() ;

      if ( (fabs(pair_mass - jpsi_mass) < window_halfwidth) &&
           (CMUO_1.em_charge() + CMUO_2.em_charge()   == 0) )
{jpsi_index += 1 ;
        JPSI_Bank new_JPSI(jpsi_index) ;          // Create an Unstored Bank
        new_JPSI.set_CMUO1_index(CMUO1.bank_number()) ;
        new_JPSI.set_CMUO2_index(CMUO2.bank_number()) ;
        new_JPSI.set_jpsi_mass(pair_mass) ;       // ...set remaining values
        record.insert(new_JPSI, record.end()) ;   // or use append() method
}   // end if pair mass is acceptable
      p_CMUO_2 = record.find_next(p_CMUO_2, "CMUO") ;
}     // end while loop over CMUO bank 2
    p_CMUO_1 = record.find_next(p_CMUO_1, "CMUO") ;
}       // end while loop over CMUO bank 1
  jpsi_total += jpsi_index ;
}         // end while(forever) loop over records in file
```