The structured data bank system for E391-library.

M. Yamaga*
High Energy Accelerator Research Organization (KEK)

March 19, 2003

Abstract

This document describes the structured data bank system in the E391-library which was introduced as an extention of the MIDAS-bank system for the KEK PS-E391a experiment.

*electric address : yamaga@post.kek.jp

Contents

1 Introduction **3**

2 Bank manager system **3**

2.1 Wrapper code as a C++ class 3

2.2 Table Definition File 4

2.3 Header Files 4

3 Example code **6**

4 Creating your own Bank manager **7**

1 Introduction

The present DAQ system of the E391a experiment is based on the MIDAS (Maximum Integrated Data Acquisition System)[1] which was originally developed for the experiment at PSI and TRIUMF. Accordingly, the raw-data format of the DAQ front-end system is so-called 'MIDAS-bank' format. The MIDAS library provide us many functionalities to access the MIDAS-bank format data. However, sometimes we would like to use rather complicated data structure in our analysis stage which is not easily achieved only by the present MIDAS library functions. For example, suppose you would like to access the calorimeter information in your physics analysis. The one channel of the calorimeter provide us many informations simultaneously such as the energy-deposit, signal-timing and detector position. Usually the energy is measured as the ADC value, and the timing is measured as the TDC value. Since ADC and TDC are measured by the different electronics module, raw-data for them are stored in the different MIDAS-bank. Thus you have to combine the data from more than one MIDAS-bank every time when you would like to obtain the whole information of a channel of the detector, which is not essential in your analysis and that might be bothering for you. In such a case, it is convenient to hold such a data in a combined data structure, such as the `struct` in the programming language C, e.g.,

```
typedef struct {
  double  adc;
  double  tdc;
  double  x;
  double  y;
  double  z;
  ...
} DETECTOR_CHANNEL;
```

Unfortunately, the MIDAS-bank system is not so convenient to handle this kind of structured data. Thus the E391-library[2] introduced the wrapper code to the MIDAS-bank system so that we could hold such a structured data in it.

2 Bank manager system

2.1 Wrapper code as a C++ class

The wrapper code for the extension of the MIDAS-bank system consists of the `C++ class` and the `C struct`. Thus the wrapper code is only accessible from the `C++` program. The `class` is derived from the `STL vector class` which holds the `struct` object as a data type. Let's call this `class` a '`Bank manager class`'. The `class` has additional functionalities to get data from the bank, to save data to the bank, and to show the contents of the bank in a certain format.

The program code of the `Bank manager class` for each data structure is automatically generated from the template of the structure before the compilation of the code in order to create them easily and safely. Only the user has to do is to prepare the template file for the structure with a certain format in a certain location. The

template file which has the suffix `.tdf` (Table Definition File) is converted to a set of the header file of `C++`, `.h` and `.icc`. All the functions of the `class` are defined as inline functions in the header files.

2.2 Table Definition File

The table definition file must have the format as follows.

```
STRUCT_NAME (TABLE_NAME:version)
{
  data_type     variable_name;  // comment
  ...
}
```

The `STRUCT_NAME` is the name of the `C struct` which will appear in the header file. It is recommended to use capital (uppercase) letters for the name of the `C struct`.

The `TABLE_NAME` is the name of the MIDAS-bank to be saved. The length of the `TABLE_NAME` is limited to be four characters from the MIDAS-bank system.

The `version` is the version number of the bank. The default version number is zero if no number is specified here. This version number is recorded in the bank as well as the data, and is used when reading the data to distinguish the bank structure which has the same MIDAS-bank name but has different structure by the update of the code.

The '{' and '}' must appear in a separate line. The description between the '{' and '}' will appear in the `C++` header file as is. Thus these lines must follow the standard `C++` syntax.

The data type which can be used in the `C` program such as `char`, `int`, `float`, ..., can be usable as a '`data_type`' in the table definition. The fixed-length array of these data type is also available by specifying the array index in the `variable_name` in the standard `C++` syntax, e.g., `int multihit[16];`.

The `STL vector` (`std::vector<>`) and `STL list` (`std::list<>`) are also acceptable as a `data_type` in order to hold the variable length data. However, the data container for the `vector` and `list` must be a simple type such as `char`, `int`, `float`, ... No `class` object, no `struct` object, no array is allowed for the container of the `vector` and `list`.

The table definition file must be located in the same directory as the usual header file in the package.

2.3 Header Files

The table definition file is converted to the `C++` header files during the build procedure of the E391-library. The command 'gmake table-headers' in the build procedure will issue the command '`gentableheader.pl`' which is a perl script to do this conversion.

The `filename.tdf` will be converted to two files, `filename.h` which describes the definition of the `struct` and `class` for the bank, and `filename.icc` which describes the inline functions of the `class`. The class has the name `Struct_name_Manager`, which starts with the capital letter followed by a small letter for the same name

as the STRUCT_NAME, followed by _Manager. Following is the example header file generated by 'gentableheader.pl' commmand.

```
// GENERATED BY gentableheader.pl version 1.0
// DATE : Tue Mar 18 20:22:36 JST 2003
//
// DO NOT EDIT THIS FILE.

#ifndef STRUCT_EXAMPLE_H_INCLUDED
#define STRUCT_EXAMPLE_H_INCLUDED

#include <vector>
#include "midas.h"
#include "MidasManager/MidasBank.h"
#include "e391bank/e391bank.h"

// Bank manager
#define STRUCT_NAME_VERSION 0

typedef struct{
  unsigned int  variable_name;  // comment
  int  thisID;
} STRUCT_NAME;
static STRUCT_NAME  struct_name_dummy;

class Struct_name_Manager
  : public std::vector<STRUCT_NAME> {
public:
  Struct_name_Manager( void* pevent )
    : m_pevent( pevent ), m_version(STRUCT_NAME_VERSION){}
  Struct_name_Manager( MidasBank* bk )
    : m_pevent( bk->eventBuffer() ), m_version(STRUCT_NAME_VERSION){}
  ~Struct_name_Manager(){}
  // extractor
  inline const void*  event_buffer() const { return m_pevent; }
  // method
  inline void  event_buffer( void* p ) { m_pevent = p; }
  inline int   get_event();
  inline int   save_event();
  inline void  show();

private:
  void*  m_pevent;
  int    m_version;
};


// inline
```

```
#include "./struct_example.icc"

#endif // STRUCT_EXAMPLE_H_INCLUDED
```

In the definition of the `struct STRUCT_NAME`, a variable '`int thisID;`' is automatically appended, which is used to check the data corruption internally in the `class`.

The function `get_event()` and `save_event()` is provided to read and write the data into MIDAS-bank system.

3 Example code

The table definition files in the officially-released E391-library is located under `${E391_TOP_DIR}/share/tables/` directories. The converted header files are located under `${E391_TOP_DIR}/include/` directories.

One can find the example analysis code with the bank manager in the package `${E391_TOP_DIR}/examples/sampleana4`. Here is the brief explanation about the code of `user_ana.cc` file.

```
  1  // -*- C++ -*-
  2  //
  3  // user_ana.cc
  4  // Template source file
  ...
 30  #include "gsim-general/GsimBank.h"
 31
 32  extern MidasBank      mbank;
  ...
173      // Generated Gamma
174      Gen_particle_Manager  gpmgr( &mbank );
175      gpmgr.get_event();
  ...
179      for( std::vector<GEN_PARTICLE>::iterator i=gpmgr.begin();
180           i!=gpmgr.end(); i++ ){
181        Hep3Vector  v( i->vx,i->vy,i->vz );
182        Hep3Vector  p3( i->px,i->py,i->pz );
  ...
218      }
```

The line#30 includes the header file for the table definition written in the `GsimBank.h` file. The original table definition is as follows.
(`$(E391_TOP_DIR)/src/sim/gsim/gsim-general/gsim-general/GsimBank.tdf`)

```
GEN_PARTICLE (MGEN:0)           // Particle information
{
  double  vx;                   // vertex (vx,vy,vz)
  double  vy;                   //
```

```
  double  vz;                   //
  double  px;                   // momentum(px,py,pz)
  double  py;                   //
  double  pz;                   //
  double  ek;                   // kinetic energy
  double  mass;                 // invariant mass
  double  time;                 // genarated time
  int     pid;                  // GEANT particle ID (KL=10,..)
  int     track;                // track# in GEANT3
  int     mech;                 // Generated mechanism (DECAY,PAIR,..)
  int     status;               // status
  int     mother;               // index number (thisID) of mother particle
  std::vector<int>  dalist;     // List of index number of daughter particles
}
```

The line#32 is needed to give the pointer to the MIDAS data bank to the constructor of the `Gen_particle_Manager class` at the line#174. You must have this description.

The line#174 creates the instance of the `Gen_particle_Manager class`.

The line#175 searches the MIDAS bank '`MGEN`' in the event data and store the contents into the `Gen_particle_Manager class`.

From the line#179 to line#218 are the example code to access the variables in each entity of the `struct GEN_PARTICLE`.

4 Creating your own Bank manager

You can create and use your own Bank manager. Suppose that you already have the local working directory with E391-library. First, you have to create the table definition file and locate it into the header-files' directory of your package. Then, go to the `${MY_TOP_DIR}/src` directory and issue the command '`gmake table-headers`'. If the conversion from the table definition file to the header file is successfully done, the following message will appear.

```
Generating ./struct_example.[h,icc] with STRUCT_NAME : STRC ver#0 : cla
ss Struct_name
```

The command '`gmake table-headers`' works only once at the first time you issue it. After generating the header files, you must issue the command '`gmake install-headers`' to install the generated headers to the common include directory. Then issue the command '`gmake`' to build your package. Now you can use your Bank manager in your program.

The order of the functions to be issued in order to save your Bank manager is as follows.

1. Open the MIDAS file.
2. Loop over the event.
3. Create the instance of the Bank manager.
4. Fill the data of the Bank structure and push_back it to the Bank manager.

5. Issue the `save_data()` function of the Bank manager.
6. Issue the `writeEvent()` function of the MIDAS file.
7. Go to the next event (3)

The example code can be found in the file:
`${E391_TOP_DIR}/src/sim/gsim/gsim-general/src/GsimMain.cc`

References

[1] http://midas.psi.ch

[2] M.Yamaga, E391-library manual, E391a TechNote.