TupleManager in E391-library

M. Yamaga*
High Energy Accelerator Reserch Organization, KEK,
1-1 Oho, Tsukuba, Ibaraki 305-0801, Japan

March 24, 2003

Abstract

This document describes the software tool 'TupleManager' which is implemented as a wrapper class of the HBOOK functions in the E391-library.

*email : yamaga@post.kek.jp

1 Introduction

HBOOK[1] and PAW[2] is the powerful tools for the physics analysis. Like many other code in the CERN library, most of the HBOOK functions are written in FORTRAN. On the contrary, most of the E391-library[3] code are written in C++. Calling the FORTRAN functions from the C++ programs is somehow complicated and bothering for the code developer.

TupleManager was prepared as a wrapper class in the E391-library to access the HBOOK functions of the CERN library from the C++ program. It provides us the easy way to create, read and write the histograms and ntuples without calling the FORTRAN subroutines from C++ by yourself. This class, however, only have the minimal set of the functions for the manipulation of the histograms and ntuples. It is still growing up, and all user should contribute to the development of this class so that it would become more convenient.

2 TupleManager class

TupleManager is the basic class to manipulate the input/output of the RZ file, and hold the list of the histograms and ntuples to be read from/written to the file.

This section describes each function in the TupleManager class.

2.1 Requirement

You need to include the header file to use TupleManager in your program.

```
#include "TupleManager/TupleManager.h"
```

You also need to add the TupleManager library to the library list in the Makefile. The name is libTupleManager.a for the static link, and libTupleManager.so for the dynamic link. Either can be specified by the description, `-lTupleManager`, in the Makefile. You will also have to append the CERN library, `-lpacklib`, in the library list.

2.2 Initialization

```
void  TupleManager::init( int hlimit );
```

hlimit	:	NWPAW (i.e. maximum size of the PAW common block)
return	:	Nothing

HBOOK initialization function. It calls HLIMIT in it.

```
void  TupleManager::initMap( int hlimit, const std::string& mapname );
```

hlimit	:	NWPAW (i.e. maximum size of the PAW common block)
mapname	:	Name for the HLIMAP to distinguish the shared memory map.
return	:	Nothing

HBOOK initialization function for the shared memory. It calls HLIMAP in it.

2.3 Constructor/Destructor

```
TupleManager();
~TupleManager();
```

No parameter is required for them.

The destructor calls the `TupleManager::close()` if the opened file was not closed yet.

2.4 Open an RZ file

```
int  ropen( const std::string& fname, const std::string& pawdir, int lun );
```

fname : Filename to be opened.
pawdir : PAW directory name to hold the histograms and ntuples.
lun : Logical unit number for the FORTRAN open() call.

return : ISTAT variable of HROPEN function. (0:success, other:failed)

Open an RZ file to read the contents. It issues the HROPEN with read-only option in it.

```
int  wopen( const std::string& fname, const std::string& pawdir, int lun );
```

fname : Filename to be opened.
pawdir : PAW directory name to hold the histograms and ntuples.
lun : Logical unit number for the FORTRAN open() call.

return : ISTAT variable of HROPEN function. (0:success, other:failed)

Open an RZ file to write the histograms and ntuples in it. It issues the HROPEN with create option in it. The existing file will be deleted and replaced with new contents.

2.5 Close an RZ file

```
int  close();
```

No parameter.
return : QUEST(1) variable after the HREND function. (0:success, other:failed)

Close the RZ file. If the file was opened to write, `TupleManager::write(0)` is automatically called in this function to dump all histograms and ntuples.

2.6 Write individual histogram into a file

```
int  write( int hid = 0 );
```

hid : Histogram ID to be written. 0 means all the histograms and ntuples.

return : QUEST(1) variable after the HROUT function. (0:success, other:failed)

Dump the histograms and ntuples specified by `hid` in the current PAW directory.

2.7 Create a CWN

```
TMNtuple*  ntuple( int hid, const std::string& title,
                   const std::string& block,
                   void* vars, const std::string& varname );
```

hid	:	Ntuple ID to be created.
title	:	Ntuple title.
block	:	Block name to be created.
vars	:	Pointer to the first variable for the block.
varname	:	Character array to specify the variables in the block.
return	:	Pointer to the created ntuple.

Create a new CWN and add it to the ntuple list in the TupleManager.

2.8 Define a CWN to be read

```
TMNtuple*  ntuple( int hid, const std::string& block, void* vars );
```

hid	:	Ntuple ID to be created.
block	:	Block name to be created.
vars	:	Pointer to the first variable for the block.
return	:	Pointer to the defined ntuple.

Define a new CWN and add it to the ntuple list in the TupleManager. This function is only available to read the contents of CWN from the RZ file.

2.9 Create a RWN

```
TMNtuple*  ntupleR( int hid, const std::string& title,
                    void* vars, const std::string& varname );
```

hid	:	Ntuple ID to be created.
title	:	Ntuple title.
vars	:	Pointer to the first variable.
varname	:	Character array to specify the variables.
return	:	Pointer to the created ntuple.

Create a new RWN and add it to the ntuple list in the TupleManager. This form requires the `vars` variable, which is usually the array of float variable, or the `struct` that consists of the float variable.

```
TMNtuple*  ntupleR( int hid, const std::string& title,
                    const std::string& varname );
```

hid	:	Ntuple ID to be created.
title	:	Ntuple title.
varname	:	Character array to specify the variables.
return	:	Pointer to the created ntuple.

Create a new RWN and add it to the ntuple list in the TupleManager. This simple form requires only the character array to specify the variables as the parameter. The actual memory to store these values are allocated automatically by the TupleManager class.

2.10 Define a RWN to be read

```
TMNtuple*  ntupleR( int hid, void* vars );
```

hid	:	Ntuple ID to be defined.
vars	:	Pointer to the first variable.
return	:	Pointer to the defined ntuple.

Define a new RWN and add it to the ntuple list in the TupleManager. This function is only available to read the contents of RWN from the RZ file.

2.11 Create a 1-D histogram

```
TMHistogram*  histogram( int hid, const std::string& title,
                         int xbin, double xmin, double xmax );
```

hid	:	Histogram ID to be created.
title	:	Histogram title.
xbin	:	Number of bins in x-axis.
xmin	:	Minimum value of the x-axis.
xmax	:	Maximum value of the x-axis.
return	:	Pointer to the created histogram.

Create a new 1-D histogram and add it to the histogram list in the TupleManager.

2.12 Create a 2-D histogram

```
TMHistogram*  histogram( int hid, const std::string& title,
                         int xbin, double xmin, double xmax,
                         int ybin, double ymin, double ymax );
```

hid	:	Histogram ID to be created.
title	:	Histogram title.
xbin	:	Number of bins in x-axis.
xmin	:	Minimum value of the x-axis.
xmax	:	Maximum value of the x-axis.
ybin	:	Number of bins in y-axis.
ymin	:	Minimum value of the y-axis.
ymax	:	Maximum value of the y-axis.
return	:	Pointer to the created histogram.

Create a new 2-D histogram and add it to the histogram list in the TupleManager.

3 TMNtuple class

TMNtuple is the container class to manipulate the ntuple.

3.1 Header file

You need to include the header file to use TMNtuple in your program.

```
#include "TupleManager/TMNtuple.h"
```

3.2 Constructor/Destructor

```
TMNtuple( const TupleManager* );
~TMNtuple();
```

These functions usually called by the TupleManager class. User should not call these functions directly from user program.

3.3 Booking function

```
// CWN
void  bookCWN( int hid, const std::string& title,
               const std::string& block,
               void* vars, const std::string& varname );
// CWN, read only
void  bookCWN( int hid, const std::string& block, void* vars );

// RWN by vars
void  bookRWN( int hid, const std::string& title,
               void* vars, const std::string& varname );
// RWN by name
void  bookRWN( int hid, const std::string& title,
               const std::string& varname );
// RWN by vars, read only
void  bookRWN( int hid, void* vars );
```

These functions usually called by the TupleManager class. User should not call these functions directly from user program.

3.4 Add variables to a CWN

```
void  addVars( const std::string& block,
               void* vars, const std::string& varname );
```

block	:	Block name to be added.
vars	:	Pointer to the first variable to be added.
varname	:	Character array to specify the variables in the block.
return	:	Nothing.

CWN can handle more than 50 variables in principle. However, the function HBNAME which is called in the booking function of ntuple only accepts up to 50 variables at a time. Multiple call of HBNAME allows user to append the variables to the CWN definition in order to use more than 50 variables. The `addVars` calls HBNAME in it so that user can add such variables.

3.5 Fill a RWN variable by name

```
void  fill( const std::string& vname, float val );
```

vname : Variable name to be filled.
val : Value to be filled.

return : Nothing.

Fill the variable data of RWN which is created with the variable names only.

3.6 Dump the data buffer to the output file

```
int  dump();
```

No parameter.

return : QUEST(1) variable after the HFNOV,HFNT or HFN function.
(0:success, other:failed)

Dump (write) the data buffer to the output file. HFNOV (RWN on shared-memory), HFNT (CWN) or HFN (RWN) is called in it, respectively.

3.7 Clear the RWN data buffer

```
void  clear();
```

No parameter.

return : Nothing.

Clears the data buffer of RWN which is automatically allocated. Clear means the variables are filled with zero. It is only valid if the RWN is created with the variable names only. In other type of ntuple, this function does not work, and clearing the data buffer is the users responsibility.

3.8 Read contents of the file

```
int   read( int eventNo );
```

eventNo : Event number to be read.

return : IERR of the HGNT,HGNTF or HGNF function.
(0:success, other:failed)

Read the ntuple contents of the specified event number. All blocks are read if the ntuple has multiple blocks.

3.9 Total number of entries of the ntuple

```
int   totalEntries();
```

No parameter.

return : Total number of entries of the ntuple.

Return the total number of entries of the ntuple. It calls HNOENT in it.

4 TMHistogram class

TMHistogram is the container class to manipulate the histogram.

4.1 Header file

You need to include the header file to use TMHistogram in your program.

```
#include "TupleManager/TMHistogram.h"
```

4.2 Constructor/Destructor

```
TMHistogram( const TupleManager* );
~TMHistogram();
```

These functions usually called by the TupleManager class. User should not call these functions directly from user program.

4.3 Booking function

```
void  book( int hid, const std::string& title,
            int xbin,double xmin, double xmax );
void  book( int hid, const std::string& title,
            int xbin,double xmin, double xmax,
            int ybin,double ymin, double ymax );
```

These functions usually called by the TupleManager class. User should not call these functions directly from user program.

4.4 Fill the histogram

```
void  fill(  double x, double weight=1 );
void  fill1( double x, double weight=1 );
void  fill(  double x, double y, double weight );
void  fill2( double x, double y, double weight=1 );
```

x : x-value to be filled.
y : y-value to be filled (valid for the 2-D case).
weight : Weight of the data.

return : Nothing.

Fill the histogram by HFILL.

5 Example code

Example code can be found in E391-library.

`${E391_TOP_DIR}/examples/sampleana[1234]/src/*.cc`

6 Development of the TupleManager

TupleManager is provided as a package of the E391-library. You can extend the TupleManager class by yourself in your local directory.

When compiling the source code of TupleManager class, they require the special header files, `ccfortran.h` and `ccfortran.icc`, which is also a package of the E391-library. These header files provide the wrapper code of each FORTRAN subroutine of CERN library. For now they have only the minimal set of the FORTRAN subroutines. If you need to use other functions in the CERN library in your program, you should ask the code manager to add new wrapper code there.

7 Contact

Questions, comments, suggestions or requests should be sent to :


```
yamaga@post.kek.jp (Mitsuhiro YAMAGA)
kensh@post.kek.jp (Ken SAKASHITA)
```


References

[1] HBOOK reference manual.

[2] PAW - Physics Analysis Workstation - An Introducroty tutorial.

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