CDF/DOC/COMP_UPG/PUBLIC/5543

R. St.Denis

January 16, 2001

CDF Offline User's Guide

The Guide

CDF PHYSICS ANALYSIS PACKAGE

Offline Version 3.11.0

Authors:

R. St. Denis

With contributions from[1]:

[1] A list of relevent web pages and CDF notes is found in Appendix A.

Contents

1 Introduction **1**

2 Getting Started **3**

3 User Methods **5**

3.1 General Comments 5

3.1.1 Name conventions 5

3.1.2 Including CDF Offline features in C++ code 6

3.1.3 AC++ Framework parameters 7

3.1.4 Method return values 7

3.1.5 Exchange of your own storable objects 7

3.2 Managers 8

3.3 Records Available 8

3.4 User Initialization 9

3.5 Event analysis method 10

3.6 User termination method 11

3.7 Other User Methods 12

3.7.1 Begin Run 12

3.7.2 Other 13

3.7.3 End Run 14

3.7.4 Abort Job 14

3.7.5 Clone 15

3.8 The Module Constructor and Destructor 16

3.8.1 Constructor 16

3.8.2 Destructor 17

4 TCL files **18**

4.1 General Information on TCL Files 18

4.2 Declaration of Talk-to Parameters 19

4.2.1 Declaration of the parameter: General type 19

4.2.2 Definition of the Command to Set the Parameter 20

4.2.3 Declare to the Framework 21

4.2.4 Description of the Parameter 21

4.3 Input/Output 22

4.3.1 CDF file types 22

4.3.2 Other CDF file types 22

4.3.3 Input datasets 22

4.3.4 Run / event selection 26

4.3.5 Listing Status of I/O 27

4.3.6 Output files 27

4.4 AC++ Framework Commands 31

4.4.1 Basic commands at the AC++ Prompt 31

4.4.2 events 31

4.4.3 filter 32

4.4.4 show 32

4.4.5 path 32

4.4.6 module 35

4.4.7 sequence 35

4.4.8 creator . . . 37

4.4.9 AC++ short commands . . . 37

4.4.10 Generic commands in any talk-to . . . 38

4.5 AC++ Required Module Commands . . . 38

4.5.1 Decompression of Data as they are Read (Puffing): PuffModule . . . 38

4.5.2 ErrorLoggerManager . . . 39

4.5.3 CalibrationManager . . . 41

4.5.4 GeometryManager . . . 42

4.5.5 SignalManager . . . 44

4.6 Histograms: . . . 45

4.7 Magnetic field . . . 45

4.8 Beam position for MCarlo . . . 45

4.9 AC++ Example . . . 47

5 Creating Histograms and Ntuples **50**

5.1 Booking and Filling Histograms/Ntuples . . . 50

5.1.1 Book a 1−dimensional histogram . . . 51

5.1.2 Book a 2−dimensional histogram . . . 52

5.1.3 Book a Profile histogram . . . 53

5.1.4 Book an Ntuple . . . 53

5.1.5 Define the Columns of an Ntuple . . . 54

5.1.6 Clear the Ntuple . . . 55

5.1.7 Filling a Histogram: Accumulate Step . . . 55

5.1.8 Filling an Ntuple: Capture Step . . . 56

5.1.9 Filling an Ntuple: Store Step . . . 56

5.1.10 Sample AC++ program to book and fill histogram, Ntuple . . . 57

5.2 Histogram output – the TCL file 61

A Useful References **62**

B "Getting Started" Failures **63**

B.1 *gmake* Trouble 63

B.2 PAW Trouble 64

C Program Structure **67**

D Full Example of an Analysis Module **69**

D.1 Description of the Directory Structure 69

D.1.1 The Head of the Release 69

D.1.2 Packages 70

D.2 The Header File 70

D.3 The C++ File 73

D.4 The Build File 77

D.5 The Makefile 79

E Advanced Features of the CalibrationManager Required Module **81**

F Process Names **83**

G Calibration Status List **88**

H Advanced Parameters in Module Talk-To's **89**

H.1 ENUM 89

H.2 List of Parameters 90

Chapter 1

Introduction

The CDF Physics Analysis package AC++[1] is intended to provide a C++ based framework for physics analysis with access to the Offline Code Repository. Although all CDF data types can be processed with AC++ the program is designed primarily for analysis of physics output files (PADS), and (later) of DSTs and MINIs. These files may be in the EDM[2], TRYBos[3] or, for silicon, ASCII format. All event input/output is done by AC++ – the user has to provide only the name(s) of the input/output data set(s) and the rest is provided for by the data handling module. AC++ also provides access to physical variables (e.g., momentum, energy), so the user can write physics analysis programs without detailed knowledge of the CDF data structure (EDM). An extensive set of utility routines (e.g., kinematics, event shape, secondary vertex finding, b-tagging, etc.) is available as part of the CDF Offline package.

The basic program structure (Appendix C) is extremely simple. Three C++ methods are normally supplied by the user: job initialization, event processing, and job termination (see Ch. 3). Reconstructed objects (tracks, vertices, cal. objects) can be accessed with C++ iterators[4]. For Monte Carlo generated events, the MC "truth" information is accessible in the same way as reconstructed tracks and vertices (see Ch. **??**).

The AC++ framework allow users to share their basic C++ methods which in fact belong to a C++ class called a *module*. Since most analyses are driven by some number of parameters, the AC++ framework allows for a uniform syntax to set parameters according to a file given as input to the program. The file is written in TCL syntax and is referred to as *the TCL file*. Parameters specific to a module are referred to as *"talk-to"* parameters and one can "talk-to" the module in order to change its behaviour. Any program (binary) produced may be run and all commands in the TCL file can be entered interactively.

Advanced features in the AC++ framework allow for modules to be run in any order in a *path*, to allow for multiple analysis paths, to allow for analysis in a path to be stopped by a *filter* module, and to allow multiple *streams* of output to be assigned to different paths. Control of the order, path definitions and output streams is specified in the TCL file.

[1] `http://www-cdf.fnal.gov/upgrades/computing/projects/framework/frame_over/frame_over.html`
[2] `http://www-cdf.fnal.gov/upgrades/computing/projects/edm/edm.html`
[3] `http://www-cdf.fnal.gov/upgrades/computing/projects/trybos/trybos.html`
[4] An extension of the DO loop concept in FORTRAN.

This document describes all features of the CDF Offline and the AC++ framework. For first−time users, the important parts to read are Ch. 2 (getting started), Ch. 3 (user routines), Ch. 4 (event input), Ch. **??** (loops over tracks), and Ch. **??** (track attributes).

This package and its documentation are taken from the ALEPH collaboration who managed to make the ALPHA package a very easy to use and extremely well documented package. We are indebted to their wisdom in having put the elements of a particle physics analysis in such a clear and concise format.

Version 3.11.0	January 16, 2001 : First Release of Document with some tests done on the veracity of Chapter 2.

Chapter 2

Getting Started

This chapter gives a brief introduction to getting started. If this does not work, have a look at Appendix B for some hints.

Four files must be provided to run an CDF offline job:

1. A header file which contains the C++ declarations for the user methods (see Ch. 3).
2. A file which contains the C++ code for the user methods (see Ch. 3).
3. A file used to build the main executable consisting of the user code and other necessary offline code (see Ch. 3).
4. A TCL file which defines the input / output data files, and allows the user to select optional parameters (see Ch. 4).

The libraries needed to link the program are described in Appendix **??**.

To start up log into fcdfsgi2.fnal.gov[1]

1. Set up the cdf environment with:

   ```
   source ~cdfsoft/cdf2.cshrc
   setup cdfsoft2 3.11.0
   ```

   where 3.11.0 is the version of the offline package.

2. Create a subdirectory, which will be called *ana* here, in which to work and go to that directory:

   ```
   newrel -t 3.11.0 ana
   cd ana
   ```

   This uses Software Release Tools (SRT). For more hints and details on SRT see Appendix **??**. The specific name of the directory is arbitrary. This directory will be the base directory, with a number of subdirectories that will be used by the job.

[1]See http://www-cdf.fnal.gov/offline/runii/fcdfsgi2/ to get an account and/or contact cdf_code_management@fnal.gov to get your own installation of the CDF Offline software.

3. Set up the analysis enviroment

   ```
   srt_setup -a
   ```

4. Add a package with a test program in it from which to build. For this guide the following package is used is an example:

   ```
   addpkg -h ExampleMyModule
   ```

5. Build the executable:

   ```
   gmake
   ```

   If this fails, you may have run out of disk space: see Appendix B.1 for hints on how to determine this and what to do.

6. If it succeeded, run the job as follows:

   ```
   rehash
   cd ExampleMyModule
   ExampleMyModule_test run.tcl
   ```

You will get a histogram of J/ψ's that can be seen with PAW. Here is a typical session:

```
>paw

 Calling 2000 version of paw-X11

 ******************************************************
 *                                                    *
 *            W E L C O M E    to   P A W             *
 *                                                    *
 *       Version 2.11/13       6 December 1999        *
 *                                                    *
 ******************************************************
 Workstation type (?=HELP) <CR>=1 :
 Version 1.26/04 of HIGZ started
 *** Using default PAWLOGON file "/home/stdenis/.pawlogon.kumac"

 the pro .pawlogon runs
 pawlogon executed for module/hybrid tests
PAW > h/file 33 appexample.hbook
PAW > cd //mymodule
PAW > h/pl 100
PAW > h/pl 101
PAW > exit
```

- If you have trouble getting PAW to display, see Appendix B.
- Each module gets its own subdirectory in PAW: go to the subdirectory with your modulename. Note that the case sensitivity is gone.

Chapter 3

User Methods

In this chapter, C++ methods which are intended to be modified by the user are described. These are the methods of the analysis *module* which is written to do some analysis job. Since the module is in fact a C++ class, the task is to write methods that will belong to this class.

Normally, only three methods have to be provided by the user: initialization (beginJob), event analysis (event), and program termination (endJob). Full examples for these three methods are in Appendix D. Other methods which may be modified by the user are also described in this chapter. Finally, a "constructor" and "destructor" can be specified for the module. Parameters that can be set by "talking-to" the module are described in the header file for the module and initialized in its constructor.

User methods should be provided as a plain C++ file containing the code and a C++ header file where the methods are declared. The description below gives the format of the method as it is declared in the header file and an example of an empty method is included in each case.

For all user methods, default versions exist in the cdf offline libraries that are loaded automatically if no user code is given.

To use the methods, they must be built together with the rest of the libraries in the CDF offline. For this, a "Build" file is required. An example of such a file is found in Appendix D.

3.1 General Comments

3.1.1 Name conventions

Classes	All C++ classes symbols defined in AC++ start with an uppercase letter and follow with uppercase letters and no underscores. **Example:** `MuonObject`.
Private Data Members	All private data members start with an underscore. **Example:** `_debug`.

Methods	All methods start with a small letter with uppercase letters demarking words **Example:** `beginRun`.

Further recommendations on coding guidelines may be found in:

`http://cdfsga.fnal.gov/computing/coding_guidelines/`

.

3.1.2 Including CDF Offline features in C++ code

In addition to methods, the Offline package contains include files which have to be included at the beginning of user methods or functions. There are a number of such includes:

The Module's header file

C header files

Header Files from Other Users' Code ("Collaborating class headers")

For the module called MyModule in a package ExampleMyModule, these sets of statements can be included as follows (see Appendix D for the description of a package and the full listing):

```
//-----------------------
// This Class's Header --
//-----------------------
#include "ExampleMyModule/MyModule.hh"

//-------------
// C Headers --
//-------------
#include <cassert>

//---------------
// C++ Headers --
//---------------

//-------------------------------
// Collaborating Class Headers --
//-------------------------------
#include "HepTuple/HepHist1D.h"

#include "Edm/EventRecord.hh"
```

```
#include "Edm/ConstHandle.hh"
#include "StorableBanks/CMUO_StorableBank.hh"
```

Important! The following sequence of statements must be observed:

1. your module's class header
2. C headers
3. the collaborating Class headers
4. your executable C++ statements

If you don't do so, you may end up with specially unintelligible C++ compiler diagnostics...

3.1.3 AC++ Framework parameters

Some parameters about the run are accessible in the framework and are described in Ch. 4.

3.1.4 Method return values

All methods of a module return one of the two values: `AppResult::OK` or `AppResult::ERROR` and are therefore declared to have return values of type AppResult.

Throughout the sections it will be noted that each method also contains the C++ *virtual* qualifier. The reason for this is a detail of the implementation of AC++ and is described in[1]. This is the mechanism by which the AC++ Framework ensures that every module has default methods and hence users are not required to write methods unless they use them.

3.1.5 Exchange of your own storable objects

Any information you wish to store or communicate to another module, must be put in a storable object. A storable object is something that must have method defined for how the output is to be formatted. These objects can be attached to and fetched from the event. The reason that they must be storable is that it is assumed that if you should choose to produce some output, it should be possible for that output to go to a file or to be communicated to another module. In this way, modules can access objects whether those objects come directly on input or are created by other modules.

[1] `http://www-cdf.fnal.gov/upgrades/computing/projects/framework/frame_over/frame_over.html`

If you need to create your own storable objects, you should follow the instructions in **??**.

There is an exception to this for the "memory database" and advice on when and how to use this is given in section **??**.

3.2 Managers

A user program needs a variety of manager modules, usually just called "managers", to provide the following services:

- Opening the CDF data base and reading constants and calibrations
- Updating of constants for each run
- CDF Geometry
- Error message control
- Histograms
- Event compression and decompression

A complete description of the managers is found in 4.5. For getting started and most applications, the default behaviour of the managers is sufficient. Managers for the most part should only keep the areas for which they are responsible running smoothly, and anticipate the needs without the user noticing and without need for frequent interaction.

3.3 Records Available

The data are stored in objects[2]. There are three objects that can be accessed with various data contents[3]:

AbsEvent* theJobRecord=AbsEvent::initJobRecord(); Information available from the time the program starts up.

AbsEvent* theRunRecord=AbsEvent::theRunRecord(); Information pertinent to an entire run such as calibrations and other settings

AbsEvent* theEvent=AbsEvent::theEvent(); The event record containing raw data hits, reconstructed objects, etc.

[2]For FORTRAN users:this is analogous to the way that they were stored in BOS banks or common blocks.

[3]FORTRAN users:think common block.

Information from these records is available only after the various times indicated by when they are created. For example, when one is analyzing events, it is possible to access the beginJob, beginRun and event information.

Important: It is essential to check that a record's pointer is not zero before trying to use it. Otherwise the program will core dump. Therefore, if you try to get the event pointer, check it as follows:

```
AbsEvent* theEvent=AbsEvent::theEvent();
if(theEvent!=0){
  // .. ok to use
}else{
  // .. signal and error and return
}
```

What to do with the event and how to signal errors is described later.

Calibration information is handled differently and this is described in **??**.

3.4 User Initialization

virtual AppResult beginJob(AbsEvent* theJobRecord)

beginJob is called once at the startup of the program.

Method arguments:	*AbsEvent* theJobRecord*	The record containing job information.
Default	no action: return AppResult::OK;.	

This method should be used to book histograms and to perform other user initializations.

All standard initialization work is performed automatically in the AC++ framework before the beginJob method is called. The standard AC++ initialization includes:

- Initialization of HBOOK (default : 100,000 words working space) through the HepHbookManager (See 4.6 for a description of histogram handling).
- Reading the user's TCL file (See Ch. 4 for a description of TCL files)
- Initialization of AC++

Unlike in HBOOK, there is no need to be concerned about memory allocation.

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record: same as the argument, theJobRecord.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::beginJob(AbsEvent* theJobRecord)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.5 Event analysis method

virtual AppResult event(AbsEvent* anEvent)

event is called once for each event.

Method arguments	*AbsEvent* anEvent* The record containing event information.
Default	no action: return AppResult::OK;.

The current event is read in, unpacked, and ready to be analyzed when *event* is called. This is the heart of analysis.

This is a pointer to the memory location where the event data are stored. The method arguments must be given even if they are not explicitly used. Typically they *are* used to access various bits of information.

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record.
AbsEvent::theEvent( )	The event Record: same as the argument, anEvent.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::event(AbsEvent* theEvent)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.6 User termination method

$\boxed{\text{virtual AppResult endJob(AbsEvent* endJob)}}$

endJob is called once at the end of execution.

Method arguments	*AbsEvent* endJob* The record containing information at the job's end
Default	no action: return AppResult::OK;.

This method can be used for anything which needs to be done at the end of a job: histogram manipulations, freeing memory and closing files. Histogram output is done automatically by the AC++ HepHbookManager described in 4.6.

endJob must never be called directly. For program termination, notify the framework by using the framework()->requestStop() method. An example of ths usage is:

```
if(tooBadForWords){
  framework()->requestStop();
  return AppResult::ERROR;
}
```

Upon return, AC++ will return to the $ANA >>$ prompt where *exit* can be typed.

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record.
AbsEvent::theJobRecord( )	The record at endjob: same as the endJob argument.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::endJob(AbsEvent* theJobRecord)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.7 Other User Methods

The methods in this section normally do not have to be modified. As mentioned above, default versions of all user methods are loaded if no new versions are provided.

3.7.1 Begin Run

$\boxed{\text{virtual AppResult beginRun(AbsEvent* beginRun)}}$

beginRun is called for each run at its start.

Method arguments	*AbsEvent* beginRun* The record containing begin run information OR the event record if the endRun was called because of a change in run number.
Default	no action: return AppResult::OK;.

This method is called once a new run is encountered on the event input file, i.e.,

- either a run record is read on the input file
- or the run number in an event record has changed
- or both conditions are fulfilled.

beginRun may be used to initialize run−dependent data or to print run statistics.

Input arguments	
AbsEvent* anEvent	This is the begin run record.
Default	no action: return AppResult::OK;.

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record, same as beginRun argument if there was a true beginRun event; otherwise this is null.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::beginRun(AbsEvent* theRunRecord)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.7.2 Other

$\boxed{\text{virtual AppResult other(AbsEvent* anEvent)}}$

if it isn't beginRun, event or endRun, it's other.

Method arguments	*AbsEvent* anEvent* The record containing event information.
Default	no action: return AppResult::OK;.

This method is called for records that are not beginRun, endRun or events. Typically these are slow control records.

Input arguments

AbsEvent* anEvent	This is the record just read in.
Default	no action: return AppResult::OK;.

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record, if there was a true beginRun event; otherwise this is null.
AbsEvent::theEvent( )	The event Record: same as the argument, anEvent.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::other(AbsEvent* anEvent)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.7.3 End Run

$\boxed{\text{virtual AppResult endRun(AbsEvent* anEvent)}}$

endRun is called at the end of each run.

Method arguments	*AbsEvent* anEvent* The record containing end run information OR the event record if the endRun was called because of a change in run number.
Return value	return AppResult::OK;.

This method is called when an end run record is encountered or a run number changes. This may be used:

- to compute and store run statistics
- reset counters for the next run
- reset histograms

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record, if there was a true beginRun event; otherwise this is null.
AbsEvent::theEvent( )	The event Record: same as the argument, anEvent.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::endRun(AbsEvent* theEvent)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.7.4 Abort Job

$\boxed{\text{virtual AppResult abortJob(AbsEvent* anEvent)}}$

abortJob is called when a fatal signal is detected.

Method arguments	*AbsEvent* anEvent* Currently set to null.
Default	no action: return AppResult::OK;.

This method is called when a fatal signal like SEGV occurs. This may be used to:

- free allocated memory
- to compute statistics if you can!

Records Available:

AbsEvent::initJobRecord( )	The beginJob Record.
AbsEvent::theRunRecord( )	The beginRun Record, if there was a true beginRun event; otherwise this is null.
AbsEvent::theEvent( )	The event Record: same as the argument, anEvent.

Format in .cc file Assume that the module class is called MyModule, then the blank method looks like:

```
AppResult MyModule::abortJob(AbsEvent* anEvent)}{
   // Put code here
   // ..
   // ..
  return AppResult::OK;
}
```

3.7.5 Clone

virtual AppModule* clone(const char* cloneName);

bring in the clones

Method arguments	*const char* cloneName* The name of the clone. When a module is being used in multiple paths a new copy of the module, a clone, is made. At the time this is done, the clone is given a name. There may be multiple paths and hence multiple clones.
Default	return a pointer to the new copy of the cloned module.

Format in .cc file Assume that the module class is called MyModule, then the *full* method looks like:

```
AppModule*
MyModule::clone(const char* cloneName)
{
  return new MyModule(cloneName,"this module is a clone MyModule");
}
```

The usage of clones within the AC++ enviroment is described in Appendix **??**.

3.8 The Module Constructor and Destructor

An AC++ module is an object and hence has a constructor and destructor. The constructor needs to be explicitly specified when one wishes to allow parameters to be modified by "talking-to" the module. The destructor needs to be specified when one allocates memory for an object using the C++ "new" command and does not attach the command to an event.

3.8.1 Constructor

$\boxed{\text{MyModule::MyModule(const char* theName, const char* theDescription)}}$

Arguments:	**theName**	String name of the module. This is the string with which it is referred in AC++ in the *talk-to path* and other commands.
	theDescription	String description of the module which shows up when one uses the AC++ *show* command.
Default:	No action.	

The values for theName and theDescription should be included in the header file declaration of the constructor for the module:

```
public:
  MyModule(const char* theName="MyModule",
           const char* theDescription="My First AC++ Module");
```

Format in .cc file Assume that the module class is called MyModule, then an empty constructor looks like:

```
MyModule::MyModule(const char* theName, const char* theDescription){
   // Put code here
   // ..
   // ..
}
```

3.8.2 Destructor

```
virtual MyModule::~MyModule()
```

Arguments:	NONE
Default:	No action.

The destructor should be used to free all memory that was allocated to objects using a *new* command provided that the objects were not subsequently attached to the event.

Format in .cc file Assume that the module class is called MyModule, then an empty destructor looks like:

```
MyModule::~MyModule





(){
   // Put code here
   // ..
   // ..
}
```

Chapter 4

TCL files

In this chapter, the TCL files are described. The TCL file is used to control input and output for AC++, and is used to control many AC++ features as well as to set the parameters of the various modules linked into AC++. For completeness, all AC++ commands are listed in this chapter; some are described in more detail in other chapters. Also, as more modules are added the details of the meanings of the TCL commands for those modules must be found in the documentation for that module. Major modules are documented in this guide. The user-supplied modules must all be specified in the "Build" procedures: see Appendix D.4 for an example Build file.

4.1 General Information on TCL Files

The following rules should be followed for all entries in the TCL file.

1. A hash symbol # indicates the beginning of a comment.
2. TCL commands can be given in any order.
3. $env(EVAR) will substitute the UNIX environment variable, EVAR, into the TCL file.
4. TCL commands are case-sensitive.

TCL commands may also be used to enter your own parameters into the program. This is done through the "talk-to" for your module. For example to modify parameters in the module, "ExampleTrackAnalysis", type:

```
talk ExampleTrackAnalysis
  ptCut set 0.5
exit
```

where "ptCut" is some name of the parameter. For boolean parameters, value is "true" or "false". "t" or "f" is sufficient:

```
talk ExampleTrackAnalysis
  verbose set t
exit
```

will set “verbose” to true.

In the C++ code each parameter is an object with a `value()` method and is accessed as follows:

```
    if (pt >_ptCut.value()){
      processFurther();
    }
```

where it is assumed that the track p_t has been determined somewhere in the code above where this cut is applied.

Note that the naming convention implies that these cuts are private members of the module's class. This must always be the case as cuts visible between modules will lead to a linking nightmare. If cuts must be passed between modules, they should communicate via a Storable Object as described in **??**.

Module parameters are are declared in C++ code as described in 4.2.

4.2 Declaration of Talk-to Parameters

Defining the parameters which can be modified by “talking to” a module requires:

- Declaration of the parameter as a *private* member of the module class in the module's header file.
- Definition of the TCL command by which the parameter will be described and the default value in the module's constructor.
- Declaration of the parameter to the AC++ framework in the module's constructor by attaching it to a menu.
- Providing a “Help” description of the parameter in the module's constructor.

4.2.1 Declaration of the parameter: General type

$\boxed{\text{AbsParmGeneral}< \textit{type} > \text{\_parname}}$

This is the general declaration of any kind of parameter and must be made in the header file of the class. The *type* is the kind of parameter. Some commone examples of usage are:[1]

integer:	AbsParmGeneral<int> _version;
boolean:	AbsParmGeneral<bool> _useCorrection;
float:	AbsParmGeneral<float> _ptCut;
double:	AbsParmGeneral<double> _mass;
string:	AbsParmGeneral<string> _particleName;

More sophisticated parameter types such as lists, and enum plus their declarations and usage examples are given in Appendix H.

In addition to being declared in the header file, the parameter:

- Must have the definition of the command to set its value given in the constructor of the module as described in 4.2.2. Additionally default, minimum and maximum values may be specified.
- Must have the parameter declared to the framework (added to a menu).
- May have a helpful description added as described in 4.2.4.

4.2.2 Definition of the Command to Set the Parameter

$\boxed{\text{\_parname(``TCLcommand'', this, par-default, par-min, par-max)}}$

Define the name, default value, min value, and max value.

Arguments:	**TCLcommand**	Used to refer to in TCL file where the syntax is *TCLcommand* set < *value* >.
	this	C++ synatax refers to the module
	par-default	default value of parameter (optional).
	par-min	default value of parameter (optional).
	par-max	default value of parameter (optional).
Location:	In the constructor of the module	
Example:	`_ptCut("ptCut", this, 0.0, 0.0, 2000.0)`	

Prerequisite: The parameter *_parname* must be declared in the header file as described in 4.2.1.

In addition the parameter:

[1]The syntax is precisely as given and is that of a templated C++ class. AbsParmGeneral objects have a value (as was shown), a keyword in TCL, such as ptCut, and a text description that appears in the menu. The specific kind of parameter is just passed as an argument between the < and > signs.

- Must have the parameter declared to the framework (added to a menu) as described in 4.2.3.
- May have a helpful description added ad described in 4.2.4.

4.2.3 Declare to the Framework

commands()->append(&_parname);

Arguments:	**&_parname**	Reference to the parameter
Location:	In the constructor or beginJob() methods.	
Example:	`command()->append(&_ptCut);`	

Prerequisites

- The parameter *_parname* must be declared in the header file as described in 4.2.1.
- The parameter *_parname* must have the definition of the command to set its value given in the constructor of the module as described in 4.2.2. Additionally default, minimum and maximum values may be specified.

At any time the parameter may have a helpful description added as described in 4.2.4.

4.2.4 Description of the Parameter

_parname.addDescription(*description*)

Arguments:	**description**	String describing the parameter, gets printed when "help" is give in the TCL file or in the talk-to for that module.
Location:	In the beginJob method.	
Example:	`_ptCut.addDescription("`tSave event if there exists a track with pT above this value.”);	

Prerequisites

- The parameter *_parname* must be declared in the header file as described in 4.2.1.
- The parameter *_parname* must have the definition of the command to set its value given in the constructor of the module as described in 4.2.2. Additionally default, minimum and maximum values may be specified.
- The parameter must be declared to the framework as described in 4.2.3

4.3 Input/Output

4.3.1 CDF file types

There are several CDF data file types:

TRYBos	input/output (RAW data are written in this format)
EDM or ROOT	machine−independent input/output (all official CDF datasets other than RAW data)
SVX	Silicon Detector ASCII files from the test stands or SiDet

The CDF file type for data cannot be recognized automatically. The file type must be chosen by the choice of input module. Currently the input modules are:

YBOSInput	for TRYBos files
DHInput	for ROOT files and tapes
TsAsciiInput	for SVX teststand Ascii files

4.3.2 Other CDF file types

TCL	TCLfiles (e.g., TCL commands)
HIS	histogram files (machine−dependent HBOOK format)

Examples: On fcdfsgi2, the environment variable `VAL_DATA_DIR` points to some example input files:

YBOSInput	`b4_peds_sidet_100ev.ybs`
DHInput	`calor_testfile_CalData.root`
TsAsciiInput	`laser2961.svx`

4.3.3 Input datasets

Input data is specified by the talk-to parameters in the DHInput module. After typing

```
module talk DHInput
```

into the TCL file (or at the AC++ prompt) one then describes the input according to the following[2]:

Format *include 'type-designator-name' 'name' 'run-restriction' ...*

The '*type-designator-name*' is one of the following:

- dataset
- datasets
- fileset
- filesets
- file
- files

A dataset is related to a collection of tapes such as "MDC-2_B_Samples". A fileset is a partition of a tape on which files reside. The actual location of the tape is transparent to the user when on fcdfsgi2: the files are staged automatically if needed. When tapes and staging are not available, then the *type-designator-name* is just file and one configures the TCL to read:

```
talk DHInput
include file $env(VAL_DATA_DIR)/calor_testfile_CalData.root
exit
```

As mentioned in 4.3.2, on the CDF central computers, the UNIX environment variable VAL_DATA_DIR points to a directory of input files useful for testing and this can be interpreted in the TCL file.

The *run-restriction* is optional and has any of the following formats:

- run=*run-number*
- run> *run-number*
- run< *run-number*
- run<= *run-number*
- run>= *run-number*

[2]One may use the word *input* instead of *include* for files. It works but is discouraged.

where the *run-number* may be either in decimal or hexadecimal format. The hexidecimal format is indicated by preceding the hexadecimal number by *0x*.

Any number of 'include' commands may be given – the data are read in the order the cards are given if the files are on disk; otherwise, the data are read in so that access speed to the tapedrive is optimized and staging of available tapes is done in parallel with analysis.

Example 1

```
module talk DHInput
  include dataset "RAW generic physics trigger" run=1234
quit
```

will process all files of run 1234 beloing to the "RAW generic physics trigger" dataset.

Example 2

```
module talk DHInput
  include dataset "RAW generic physics trigger" run>=0xabcdb run<=0xabcde
quit
```

will process all files of runs 0xabcdb, 0xabcdc, 0xabcdd belonging to the dataset "RAW generic physics trigger".

Example 3

```
module talk DHInput
  include files /cdf/data*/*/ab??????.????cdef
quit
```

will process all files corresponding to the pattern. The synatax is understood that "*" indicates a wildcard that may be substituted with a string of any length and "???" indicates a pattern where exactly 3 character place holders are kept, but any character my appear in any of the three places.

The current list of input files may be seen by typing

```
talk DHInput
  show
exit
```

Finer specification of specific events within runs as well as additional ways to specify runs are given in 4.3.4.

How to specify "tapes"

```
talk DHInput
  include fileset CA3000.0
exit
```

Here, CA3000.0 is the fileset name. It corresponds to partition 0 on tape CA3000 name but you need not know that of course: all tape handling is automatically done by the data handling system.

The corresponding tape will not be read directly by AC++: the system will first stage the specified file (or several files in one go) on disk; then AC++ will read this disk file.

The fileset will have a number of files, for example:

- aa01933f.0001phys
- aa01933f.0004phys
- aa01933f.0008phys
- aa01933f.0012phys

could be some files in the fileset. The number before the decimal point is the run number in hex and the number after is the run section number where a run section has about 2000 events and represents about 30s of data taking. In this case one could expect about 6000 events on the first file since it contains three partitions. Each file should be around 1 G so that one does not have trouble on filesystems that only work with up to 32 bits.

To find the datasets and filesets of interest, use the Database Browser [3] as described in Chapter **??**. There one can use hyperlinks to go from a dataset to a fileset to the files to the run/runsection.

stage list: query on staged datasets on CDF central computers

An interactive facilty called **stage** is available on the central CDF computers to know which filesets and datasets are presently staged. This may be very useful during testing, to use already staged tapes in order to avoid the staging time which may be very long.

stage list -s	List all disk resident filesets
stage list -s *'fileset_name'*	Lists the fileset if it is on disk
stage list -f	List all disk resident files

[3] `http://www-cdf.fnal.gov/internal/upgrades/computing/database/browser/browserguide.htm`

stage list -s *'file_name'*	Lists the file if it is on disk
stage list -F	Lists all the file systems (ie. disks) available and shows space used
stage list -F *'files_system'*	Lists the space for the filesystem (ie. disk) if it exists

There are a number of other actions for which the **stage** command allows; however, most are handled by the input and output modules. *stage -h* lists the commands available.

Examples: In the database browser(see section **??** for instructions), you have determined that the filesets associated with your dataset include the fileset CA3099.0. You want to know if it is on disk so you type

```
stage list -s CA3099.0
```

and you get:

```
152 Filesets in edition 280:
```

Note that you did not get your fileset listed so in fact it is not there. But you do see that there are 152 filesets out there, so you want to know what is there. You type:

```
stage list -s
```

and you get

```
152 Filesets in edition 280:
CA3086.3     (   0.0 GBytes,    0 files) cached on disk (1 reservations)
CA3086.2     (   0.0 GBytes,    0 files) cached on disk (0 reservations)
CA3086.1     (   0.0 GBytes,    0 files) cached on disk (0 reservations)
...
```

There are 152 lines (only 3 shown here). You wonder what these are so you can go to the database browser and type in one of these filesets to find out what dataset it belongs to. You find that CA3086.3 belongs to mdc2tr.

4.3.4 Run / event selection

Specific run selection may be specified in the *include* command as described in 4.3.3. In addition, the following TCL commands may be used to select particular runs or trigger numbers for analysis. Trigger numbers are set to zero at the start of a run and increase monotonically. The events may also be selected by run section (25 seconds worth of data, about 2000 events) and run number.

RunsTrigs set 100:105 108 110:114	Select RUNs 100, 101, 102, 103, 104, 105, 108, 110, 111, 112, 113, 114.
RunsTrigs set 100(1111:1112)	Select trigger 1111 on the input file.
RunsTrigs exclude 1 5 7 11 −99	Ignore runs 1,5,7,11,12,13,14,...,99

Notes:

- The RunsTrigs command selects only those events which are sent for the processing by the AC++ modules. All events will be read in any case. So one can reject events from processing, but not from input.
- The syntax of R(T) where R=run and T=trigger follows the rule that R(*) is all triggers and is the same as R:

Short format	Full equivalent
R1:R2(T)	R1(*):R2(T)
R1(T):R2	R1(T):R2(*)
R(T1:T2)	R(T1):R(T2)
R1:R2	R1(*):R2(*)
R	R(*):R(*)

4.3.5 Listing Status of I/O

The command

```
module talk DHInput
  show include
exit
```

will show the files and filesets that are to be analyzed.

```
module talk DHInput
  RunsTrigs list
exit
```

will show the selections specified by the RunsTrigs command.

4.3.6 Output files

Event output is controlled by the DHOutput module. The data set name and options are given in the talk-to of DHOutput. The objects written are controlled by ??. To output data,

- One must define an output stream by attaching a file to a stream.
- One must specify the path of analysis to attach to the stream.
- One may set optional parameters on the output stream.

Create Stream: *output create 'stream name' 'destination'*

stream name	same as on defined in output stream cards;
destination	a file name for output or the dataset ID. If a "/" or a "." is in the name, it is assumed to be a file.

Attach Path to Stream: *output path 'stream name' 'path name' 'parameter2' ...*

stream name	same as on defined in output stream;
path name	path designating the analysis modules to be called before writing. The default path is *AllPath*. Paths are described in 4.4.5.

Set Stream parameters: *output setstream 'stream name' 'parameter1' 'parameter2' ...*

stream name	same as on defined in output stream cards; see examples of how to define a path in. A "*" can be used to address all streams.
parameter	is either a single **keyword** or expression **keyword=value**.
parameters:	(optional)
list	print out current setting for the output stream
dfc	use default Data File Catalog to fill file records. Details on how to set this up may be found in **??**.
nodfc	Do not fill any Data File Catalog file record.
anysize	No restrictions on size of output file
savecatalog	Write File Content Catalog into the file. Further details on the File Content Catalog are available [4].

[4] `http://rutpc7.fnal.gov/ratnikov/Docs/DHIOModuleReference.htm`

pathname=(char) ‘path’	Specify the pathname of directory where the data files are written.
file=(char) ‘name’	Write to the named file or its successors. If a file is larger than 1GB new files are opened and the original is closed. Sequential numbers are appeneded to *‘name’* to create this set of files (nb. fileset). The *‘anysize’* option deactivites this.
dfc=(char) ‘name’	Specify Data File Catalog named *‘name’* keeping track of the output data
flush=(int) ‘size’	Specify how often the data are to be flushed to disk as determined by the *‘size’* in kB.
size=(int) ‘size’	Specify the maximum size of an output file in Kb. Files are closed and successors are written when this size is reached. Each successor has a sequential number assigned the end of the filename.
skiptill=(int) ‘run/section’	Ignore data with *‘run‘/‘section‘* less than specified: this skips over these records whereas the Run / event selection 4.3.4 commands read the records and only provide the selected events for processing. This is useful for crash recovery.

Examples:

Example 1: Output to a File

```
output path mystream mypath
output create mystream mydirectory/myfile
```

Stream with name mystream is connected to AC++ data path mypath (paths are described in 4.4.5, streams are described in 4.3.6).

Notes:

- The slash before myfile is important. Use "./< $myfile$ >" to write a file *myfile* into current directory.
- The File Content Catalog will not be written to the data file.
- No communication with DFC or DIM is necessary
- This is safe to use without any tape staging capabilities: only files.

Example 2: Output to a File with customization of parameters

```
output path mystream mypath
output create mystream mydirectory/myfile
output setstream mystream nodfc anysize savecatalog list
```

Output stream properties are specified explicitly. Stream with name mystream is connected to AC++ data path *mypath.* Output file will be *mydirectory/myfile* without any restrictions on its size.

Notes:

- The slash before myfile is important. Use "./< $myfile$ >" to write a file *myfile* into current directory.
- The File Content Catalog will be also written to the data file.
- No communication with DFC or DIM is necessary.

Multiple streams may be defined and attached to different files and different analysis paths.

Example 3: Output to a Dataset

```
output path mystream mypath
output create mystream mydataset
output setstream mystream dfc=production_file_catalog size=1000000
output setstream mystream pathname=mydirectory list
```

Almost certainly one expected this example to describe "output to a tape" and this is essentially what this is. The point is that a dataset is *stored* on a tape – but the boundaries of the tapes should not have to concern you. So you define a dataset in the database using the tools described in **??** and then here you can write to this.

For this example, a stream with name *mystream* is connected to AC++ data path *mypath.* Data will be assigned to dataset *mydataset* and corresponding output file names will be generated. Output files will be collected in directory *mydirectory*, files will be closed when size exceeds 1Gb. FILE record will be added to the DFC specified by the name *production_file_catalog* in the iomap.txt file and files will be sent for archiving. The current setup will be printed out.

unpuffEvents (default = false) Data compression

Data are written in compressed format and need to be "puffed" as they are read in. The command

unpuffEvents set 1 suppresses the (de)compression.

More detailed control of compression ("puffing") is available from the requred module "PuffModule" described in 4.5.1

4.4 AC++ Framework Commands

Commands in AC++ are issued at the *AC++>* prompt or within the "talk-to" for an individual module. In addition to the general framework commands, all modules share some identical talk-to commands. The AC++ commands are discussed here and the generic commands applicable to any module are treated in Section 4.4.10.

4.4.1 Basic commands at the AC++ Prompt

A number of commands may be included in the TCL file that directly control the job. The AC++ manual [5] describes the framework in detail.

4.4.2 events

Command used to control the looping over events

`event begin` *[-nev <n>]* Begin processing

`event continue` *[-nev <n>]* Continue processing

Where *begin* resets the current input module to start a new event processing run, whereas *continue* continues the current sequence (presumably because it was terminated prematurely by the optional qualifier *-nev* which specifies a number of events). The *continue* command acts identically to the *begin* command if no prior *begin* command had been issued.

Further control over the events: skipping events, runs etc, is available in the talk-to of the DHInputModule as described in 4.3.4.

[5] `http://www-cdf.fnal.gov/upgrades/computing/projects/framework/frame_over/frame_over.html`

4.4.3 filter

Modify the mask used to interpret filter decisions.

`filter` *[-path <pathName>] <moduleName> on/off/veto*

This command sets the mask for a particular filter in a particular path. If a path is not specified the default, AllPath is assumed. 'On' is the default behavior if this command is given to the Framework, and means the filter's decision will be used to prematurely terminate path execution. 'Off' means the filter's decision will be ignored by the Framework in deciding how to execute the path (or sequence). Veto means the logical NOT of the filter's decision will be used.

4.4.4 show

Display the available modules, paths, and processing time for each module. This also indicates the active modules. Example output from a 'show' command is given in 4.9.

4.4.5 path

Command used to create, modify and control paths. A complete description of paths is available in[6]. A path defines the order in which modules are called. Modules not in the path are not called for each event but their begin and end run and job methods are invoked. To prevent this, the modules must be also be *disabled*.

`path help`	Show help for this command
`path list`	List current path(s)
`path create` *<PathName> <mod1> <seq1> ...*	Create a new path
`path append` *<PathName> <mod1> <seq1> ...*	Append mod(s)/seq(s) to PathName at the end of the path
`path delete` *<PathName>*	Delete a path
`path disable` *<PathName>*	Disable path(s)
`path enable` *<PathName>*	Enable path(s)
`path insert` *<PathName> <mod1> <seq1> ...*	Insert mod(s)/seq(s) into PathName at start of path

[6] `http://www-cdf.fnal.gov/upgrades/computing/projects/framework/frame_over/frame_over.html`

`path remove` *<PathName> <mod1> <seq1> ...*	Remove mod(s)/seq(s) from PathName

By default, modules are placed into the path "AllPath" which contains all modules.

Here are some examples taken in the context of the ExampleMyModule_test program:

```
AC++> path list

**** Listing of all available paths ****

* = Enabled; ! = Active


    Default (all modules) path                AllPath
    Filter?  Mask  nQuery  nPassed
*   CDF required manager sequence               ManagerSequence
*     no      on        0        0                ErrorLoggerManager
*     no      on        0        0                PuffModule
*     no      on        0        0                CalibrationManager
*     no      on        0        0                GeometryManager
*     no      on        0        0                SignalManager
*     yes     on        0        0              HepHbookManager
*     yes     on        0        0              MyModule
*     yes     on        0        0              ExampleTrackAnalysis
```

One sees that AllPath contains the modules but is disabled: there is no "*" in the first column. Now create a new path, put the CalibrationManager and MyModule into the path, and show the result:

```
AC++> path create ShortPath

AC++> path append ShortPath CalibrationManager MyModule

AC++> path list

**** Listing of all available paths ****

* = Enabled; ! = Active


    Default (all modules) path                AllPath
    Filter?  Mask  nQuery  nPassed
```

```
*   CDF required manager sequence              ManagerSequence
*     no      on        0        0               ErrorLoggerManager
*     no      on        0        0               PuffModule
*     no      on        0        0               CalibrationManager
*     no      on        0        0               GeometryManager
*     no      on        0        0               SignalManager
*     yes     on        0        0             HepHbookManager
*     yes     on        0        0             MyModule
*     yes     on        0        0             ExampleTrackAnalysis

*                                            ShortPath
    Filter?  Mask  nQuery  nPassed
*     no      on        0        0             CalibrationManager
*     yes     on        0        0             MyModule
```

The path "ShortPath" has been created and is enabled. More than one path can be enabled. This can be demonstrated by enabling AllPath as well:

```
AC++> path enable AllPath
AC++> path list

**** Listing of all available paths ****

* = Enabled; ! = Active


*   Default (all modules) path               AllPath
    Filter?  Mask  nQuery  nPassed
*   CDF required manager sequence              ManagerSequence
*     no      on        0        0               ErrorLoggerManager
*     no      on        0        0               PuffModule
*     no      on        0        0               CalibrationManager
*     no      on        0        0               GeometryManager
*     no      on        0        0               SignalManager
*     yes     on        0        0             HepHbookManager
*     yes     on        0        0             MyModule
*     yes     on        0        0             ExampleTrackAnalysis

*                                            ShortPath
    Filter?  Mask  nQuery  nPassed
*     no      on        0        0             CalibrationManager
*     yes     on        0        0             MyModule
```

4.4.6 module

Command used to control and specify modules

`mod help`	Show help for the module command
`mod disable` *<ModName>...*	Disable module(s)
`mod enable` *<ModName>...*	Enable module(s)
`mod in(put)` *<ModName>...*	Specify the Input Module
`mod list`	List current modules
`mod out(put)` *<ModName>...*	Specify the Output Module
`mod talk(To)` *<ModName>...*	Talk to module
`mod action(s)` *enable/disable <ModName>/all*	Enable/Disable actions
`mod clone` *<ModName> <new-ModName>*	Clone a module

4.4.7 sequence

Command used to create, modify and control sequences. A sequence is a set of modules that should be manipulated together. An example is the clustering and tracking of the silicon could be put into a sequence. Another example is the CDF required modules described in 4.5. These can then be added and removed from a path as a set without specifying each individual module. Sequences can also contain other sequences.

`seq append` *<SeqName> <mod1> <seq1> ...*	Append mod(s)/seq(s) to SeqName at the end of the sequence
`seq create` *<SeqName> <mod1> <seq1> ...*	Create a new sequence
`seq delete` *<SeqName>*	Delete a sequence
`seq disable` *<SeqName>*	Disable sequence(s)
`seq enable` *<SeqName>*	Enable sequence(s)
`seq help`	Show help for this command
`seq insert` *<SeqName> <mod1> <seq1> ...*	Insert mod(s)/seq(s) into SeqName at the start of the sequence
`seq list`	List current sequence(s)
`seq remove` *<SeqName> <mod1> <seq1> ...*	Remove mod(s)/seq(s) from SeqName

Here is an example of sequence manipulation using ExampleMyModule_test:

```
AC++> sequence list

**** Listing of all available sequences ****

* = Enabled; ! = Active


*   CDF required manager sequence             ManagerSequence
*     no      on        0        0              ErrorLoggerManager
*     no      on        0        0              PuffModule
*     no      on        0        0              CalibrationManager
*     no      on        0        0              GeometryManager
*     no      on        0        0              SignalManager

AC++> sequence create fudge
AC++> sequence append fudge PuffModule GeometryManager
AC++> sequence list

**** Listing of all available sequences ****

* = Enabled; ! = Active


*   CDF required manager sequence             ManagerSequence
*     no      on        0        0              ErrorLoggerManager
*     no      on        0        0              PuffModule
*     no      on        0        0              CalibrationManager
*     no      on        0        0              GeometryManager
*     no      on        0        0              SignalManager

*                                             fudge
*     no      on        0        0              PuffModule
*     no      on        0        0              GeometryManager

AC++> sequence remove fudge GeometryManager
AC++> sequence list

**** Listing of all available sequences ****

* = Enabled; ! = Active


*   CDF required manager sequence             ManagerSequence
*     no      on        0        0              ErrorLoggerManager
*     no      on        0        0              PuffModule
*     no      on        0        0              CalibrationManager
*     no      on        0        0              GeometryManager
*     no      on        0        0              SignalManager

*                                             fudge
*     no      on        0        0              PuffModule

AC++> sequence insert fudge SignalManager
AC++> sequence list

**** Listing of all available sequences ****

* = Enabled; ! = Active


*   CDF required manager sequence             ManagerSequence
*     no      on        0        0              ErrorLoggerManager
*     no      on        0        0              PuffModule
*     no      on        0        0              CalibrationManager
*     no      on        0        0              GeometryManager
*     no      on        0        0              SignalManager

*                                             fudge
*     no      on        0        0              SignalManager
*     no      on        0        0              PuffModule
```

4.4.8 creator

Specify the string to use to identify the creating process for event objects.

`creator set` <*string*>

4.4.9 AC++ short commands

Short forms of the most common AC++ commands exist and are listed here:

Short Form	Long Form
mod	module
cont	events continue
talk	module talk
seq	sequence
begin	events begin
up	exit
logout	exit
ev	events
quit	exit

4.4.10 Generic commands in any talk-to

The following commands can be used in any menu in the TCL file:

exit	Leave the current menu, module, or process
help	Bring up help text for the current context.
show	Display the value of any parameters or statistics associated with the module.
echo	Send text argument to stdout (useful in scripts). This behaves like the unix echo.
action	Command used to control module actions
verbose	Turn on verbose screen output. Usage: verbose set t/f
production	Suppress all screen output. Usage: production set t/f

4.5 AC++ Required Module Commands

A number of modules are considers "required" in order for analysis to proceed. These modules include the input and output modules, the puffers and the manager modules.

4.5.1 Decompression of Data as they are Read (Puffing): PuffModule

Decompression, or "Puffing" of data is performed automatically by the "PuffModule".

To save time,unpacking of specific objects can be stopped or a list of objects to be puffed (and no others) may be given. After issuing the TCL command

```
module talk PuffModule
```

the following commands may be used to specify a list of class name strings to be puffed if present in event:

`dontPuff set` *<class1> [class2] ...*

`dontPuff add` *<class1> [class2] ...*

`dontPuff reset`

To specify a list of 4char bank name strings to be HEX dumped if in event, the commands below are used. Note to save on typing you should not type in the _StorableBank part.

`puffOnly set` *<class1> [class2] ...*

`puffOnly add` *<class1> [class2] ...*

`puffOnly reset`

The classes are:

AL	all banks are unpacked but no coordinate sorting is done
	COT coordinates
	SVX coordinates
	ISL coordiantes
	dE/dx
	Electrons
	Muons
	Jets
	Calorimeter
	Track Fits
	NO unpacking

4.5.2 ErrorLoggerManager

The ErrorLoggerManager is the interface to the Error Logging Facility. The error logger is easily used as a replacement for *cout* in C++ programs. The logging of errors to the manager in code is done as:

```
if(reallyBadMistake){
  errlog(ELerror,"[NO_AUXRUNDATA]")
        << "SvxPedDBModule::beginRun() was unable to find a"
        << "StorableSvxTsAuxRunData object in the BOR record"
        << endmsg;
  return AppResult::ERROR;
}
```

cout is replaced by *errlog* with arguments indicating the severity of the error and a string that can be used to identify errors of that class. The rest of the message looks just like a *cout* output and the line is terminated by *endmsg* instead of *endl.*

There error severities ranging from lowest to highest are:

- ELincidental
- ELsuccess
- ELinfo
- ELwarning
- ELwarning2
- ELerror
- ELerror2
- ELnextEvent
- ELunspecified
- ELsevere
- ELsevere2
- ELabort
- ELfatal
- ELhighestSeverity

With this usage, the modules interface to infrastructure in a uniform way that allows logging files to be defined for logging and real time error display.

Control of the messaging is done within a job using the ErrorLoggerManager. These are the options availble when “talking-to” the ErrorLoggerManager:

errfile	Specify whether or not errors should additionally go to a file.

Format *errfile set <file-to-log-to>*
Default None
No file writing is done

limit Specify the maximum number of times a particular message should be output.

Format *limit set <n>.*
Default 10

severity Specify the level to supress messagaes at or below. For example if this parameter is set to ELINFO, info, sucess and incidental messages will all be supressed from output.

Format *severity set <severity-string>.*
Default WARNING

Severity levels correspond to those in the code in a clear way and are listed here:

- INCIDENTAL
- SUCCESS
- INFO
- WARNING
- WARNING2
- ERROR
- ERROR2
- NEXTEVENT
- UNSPECIFIED
- SEVERE
- SEVERE2
- ABORT
- FATAL
- HIGHESTSEVERITY

4.5.3 CalibrationManager

The CalibrationManager, manages the default calibrations. By identifying the kind of analysis with the ProcessName command, the manger uses this to go to the database to find out which calibrations are valid for the run you are analyzing. This implies of course that somebody has figured this out for you before you ran.

The CDF calibration software learns which database it should use from the manager. This is communicated by the use of *database identifiers.* Some identifiers are defined by default. New identifiers can be added as described in Appendix E.

The basic commands to control the calibrations being used are listed below. Additional commands are described in Appendix E.

ProcessName Calibration Process Name that wrote the calibration you wish to use. A list of process names and their definitions may be found in F. By convention, the process name follows the syntax

Syntax of Process Name <*Destination*>_<*Mode*>_<*Detector*>
where *Destination* describes where the calibration is to be used, *Mode* describes how the calibration was taken, and *Detector* describes to which detector the calibration applies. For example, ProcessName="PROD_BW3_SVX" indicates that this set of calibrations corresponds to something that can be used in production (PROD) for the silicon detector (SVX) when the bandwidth setting is 3 (BW3).

Format *ProcessName set* <*ProcessName*>

Default PRODUCTION_TEST

Database This identifier points to the database where the calibration administration information that relates the run being analyzed to the required validated calibrations necessary to analyze the data. The intent here is to allow a text database to be chosen where one can setup one's own list of calibrations that should be applied to the run.

Format *Database set* <*database-identifier*>

Default BLOB_JUNK

DataDB Database-ID: this is the database containing the calibration data tables.

Format *DataDB set* <*database-identifier*>

Default database_oracle_offline

LoadAll Indicates that all database drivers should be loaded from the default db-id

Format *LoadAll set* <*true-or-false*>
This is boolean and can be set to "t" or "f".

Default f

list List database identifier definitions

Format *list*

4.5.4 GeometryManager

This module is needed for CdfGeometry and has the following main menu options:

DetectorMenu	Invokes the DetectorMenu submenu
Cot	Invokes the Cot submenu
PrintMenu	Invokes the PrintMenu submenu
closeG4Geometry	Close the geometry as declared to Geant4. **Format** *closeG4Geometry set <true-or-false>* This is boolean and can be set to "t" or "f". **Default** f
uniformField	Select uniform magnetic field in CdfDetector (default false). This does not set a value for the field. **Format** *uniformField set <true-or-false>* This is boolean and can be set to "t" or "f". **Default** f

The sub menu options are as follows:

DetectorMenu: All options are set as follows:

Format *<option> set <true-or-false>*
This is boolean and can be set to "t" or "f".

Defaults are shown in parentheses.

enableB4	Enable building Svx Barrel 4 (default: f).
enableBeamBox	Enable building Svx "Beam Box" (default: f).
enableSvx	Enable building Silicon detector (default: t).
enableCot	Enable building Central outer tracker (default: t).
enableCentralMuon	Enable building central muon system <OBSOLETE> (default: f).
enableForwardMuon	Enable building forward muon system <OBSOLETE> (default: f).
enableMuon	Enable building muon system (default: f).
enableCalor	Enable building calorimeter system (default: t).
enableClc	Enable building central luminosity counters (default: f).
enableTof	Enable building time-of-flight system (default: f).
enableStripChamber	Enable building strip chamber (default: t).
enablePassive	Enable building passive geometry (default: f).

enableCPR	Enable building pre-shower radiator (default: t).
enableAll	Enable building all subsystems (default:f)

Cot: All options are set as follows:

Format *<option> set <true-or-false>*
This is boolean and can be set to "t" or "f".

Defaults are shown in parentheses.

run1CTCGeometry	Whether or not to use CTC geometry in place of COT for Run1 tracking. (default: f)
run1CTCData	In the case of run1 CTC, is this data (t) or MC (f). (default: t)

PrintMenu: All options are set as follows:

Format *<option> set <true-or-false>*
This is boolean and can be set to "t" or "f".

Defaults are shown in parentheses.

printSvx	Print geometry information for the Silicon detector. (Default: f)
printCot	Print geometry information for the Central outer tracker. (Default: f)
printCalor	Print geometry information for the calorimeter system.(Default: f)
printStripChamber	Print geometry information for the strip chamber.(Default: f)
printCPR	Print geometry information for the pre-shower radiator.(Default: f)
printAll	Call the specific print method for all enabled detector nodes.(Default: f)
printTree	Print out the whole detector tree using a generic tree walk (very verbose).(Default: f)

4.5.5 SignalManager

Utility that manages system signals

4.6 Histograms:

histfile Select the Histogram file name to be used in this job. Note that this parameter is setable from any histogramming module however it can not be altered after the first begin command when it will be opened.

Format *histfile set <fileName>*

Default HepHist.dat

createHistoFile Create the histogram file

Format *createHistoFile set <true-or.-false>* This is boolean and can be set to "t" or "f".

Default t

reclen Select the Histogram file record length. The minimum value is 512 and the max is 65536.

Format *reclen set <len>*

Default 1024

The commands used in connection with the histogram package are described in detail in Chapter 5.

4.7 Magnetic field

Magnetic field can be set to a given value for Monte Carlo studies:

```
talk GEANT3
  show            #  shows current values
  help            #  options including the following
  bmagnt list     #  shows current value of mag. f.
  bmagnt set 10.12 #  set value
exit
```

This requires that the GEANT3 module have been built into the executable. More details on this module may be found in **??**. module talk GenPrimVert .

4.8 Beam position for MCarlo

The beam position is generated by the GenPrimVert module. The object ? has methods ? (see chapter **??**) that gives beam position information for real data. They also provide effective beam positions for correctly simulating the size of the luminous region in Monte Carlo.

Arbitrary beam sizes may be simulated by means of the σ_x, σ_y, and σ_z parameters of GenPrimVert

sigma_x, sigma_y, sigma_z σ_x and σ_y are floating point numbers giving the desired sizes of the luminous region in cm, transmitted to ??. The value of σ_z , if given, is returned in ???.

Format *sigma_x set* $< \sigma_x >$

Default $\sigma_x = 0$

Format *sigma_y set* $< \sigma_y >$

Default $\sigma_y = 0$

Format *sigma_z set* $< \sigma_z >$

Default $\sigma_z = 30$

sigma_t σ_T of the primary interaction in ns. This is the accuracy assumed for the known location of the interaction as measured in time by the CLC.

Format *sigma_t set* $< \sigma_t >$

Default $\sigma_t = 2$

bunch_spacing Time in ns between bunches: nb. 132ns or 396ns.

Format *bunch_spacing set* $< bunch - spacing >$

Default 400

n_bunches Number of bunches to be simulated in one event.

Format *n_bunches set* $n - bunches$

Default 1

Examples :

1. To simulate a luminous region size of 120 x 7 microns that is 36cm long:

```
talk GenPrimVert
  sigma_x set 0.0120
  sigma_y set 0.0007
  sigma_z set 36.0
exit
```

2. To smear the beam position uncertainties as measured in time, choose 150ps and this gives an uncertainty of about 10cm in z.

```
talk GenPrimVert
  sigma_t set 0.150
exit
```

3. To specify the 3 bunches interacting with 132ns time between the bunches:

```
talk GenPrimVert
  nbunches set 3
  bunch_spacing set 132
exit
```

4.9 AC++ Example

Referring again to the example given in Ch. 2, the command

```
ExampleMyModule_test run.tcl
```

was issued at the UNIX prompt and then at the AC++ prompt, the command

```
ev begin -nev 10
```

was given to process 10 events. Then the show command was given

```
show
```

and the result was:

```
Value of creator for module AC++ is UNKN

**** Listing of all available modules ****

* = Enabled; ! = Active

*   ErrorLoggerManager    Interface to the Error Logging Facility
*   PuffModule            Selectively restores the transient event data
*   CalibrationManager    Manage the default calibrations
*   GeometryManager       Module needed for CdfGeometry
*   SignalManager         Utility that manages system signals
*   HepHbookManager       Initializes HepTuple with Hbook
*   MyModule              Example user analysis
*   ExampleTrackAnalysis  Example user analysis

Input Modules
```

```
*   DummyInput             dummy input module
*   FileInput              Default Input Module
*!  DHInput                Data Handling Input Module

Output Modules
*   DummyOutput            dummy output module
*   FileOutput             Standard File Output Module
*!  DHOutput               Data Handling General Output Module

**** Listing of all available paths ****

* = Enabled; ! = Active


    Default (all modules) path                AllPath
    Filter?  Mask  nQuery  nPassed
*   CDF required manager sequence               ManagerSequence
*     no      on       0        0                 ErrorLoggerManager
*     no      on       0        0                 PuffModule
*     no      on       0        0                 CalibrationManager
*     no      on       0        0                 GeometryManager
*     no      on       0        0                 SignalManager
*     yes     on       0        0               HepHbookManager
*     yes     on       0        0               MyModule
*     yes     on       0        0               ExampleTrackAnalysis

**** Listing of all available sequences ****

* = Enabled; ! = Active


*   CDF required manager sequence             ManagerSequence
*     no      on       0        0               ErrorLoggerManager
*     no      on       0        0               PuffModule
*     no      on       0        0               CalibrationManager
*     no      on       0        0               GeometryManager
*     no      on       0        0               SignalManager

**** Execution Times for all Modules Run so Far ****

During Event Processing:
========================
Module name:  # Calls:     Mean cpu time:        Mean clk time:   Total Cpu:
----------------------------------------------------------------------------
CalibrationMana 10  0.000000+/-0.000000   0.000052+/-0.000002   0.000
DHInput         22  0.066364+/-0.055761   0.069799+/-0.058549   1.460
```

```
DHOutput        10  0.000000+/-0.000000   0.000044+/-0.000000   0.000
ErrorLoggerMana 10  0.000000+/-0.000000   0.000041+/-0.000000   0.000
ExampleTrackAna 10  0.000000+/-0.000000   0.000712+/-0.000008   0.000
GeometryManager 10  0.000000+/-0.000000   0.000041+/-0.000000   0.000
HepHbookManager 10  0.001000+/-0.001000   0.000309+/-0.000034   0.010
MyModule        10  0.001000+/-0.001000   0.000445+/-0.000005   0.010
PuffModule      10  0.014000+/-0.001633   0.016834+/-0.002022   0.140
SignalManager   10  0.000000+/-0.000000   0.000040+/-0.000000   0.000
```

- The list of available modules includes the CDF Requried Modules:
  - ErrorLoggerManager
  - PuffModule
  - CalibrationManager
  - GeometryManager
  - SignalManager
  - HepHbookManager

  plus the two modules we have added in:
  - MyModule
  - PuffModule

  The asterisks, "*", indicate which modules are enabled. This means that their begin and end run and job methods are invoked.

  For the Input and Output modules, there is only one that can be active and this is indicated by the exclamation mark, "!".

- The next table shows the default path, AllPath, which consists of the Manager Sequence, the HepHbookManager, MyModule and ExampleTrackAnalysis. This illustrates that a sequence is a group of modules that belong logically together and can be handled in paths as one object. Also, the table shows that some of the modules can filter as indicated by a "yes" in the "filter" column. The display also indicates if the filter mask is on as well as the number of attempts to filter and the number of successes for each module.

- The sequences are then listed with the same information regarding filtering as was shown for paths.

- Finally, the execution time statistics are shown for all modules. These statistics include the number of times the module was called, the mean CPU, the clock time and the total cpu time. All numbers are given for the machine upon which the job is run.

Chapter 5

Creating Histograms and Ntuples

The standard histogram package in CDF is HepTuple. This works with either HBOOK or ROOT; however, working with GEANT at the same time leads to subtle problems and to overcome these, consult **??**. If you don't want to use HepTuple the only system routines which are called automatically and which refer to HBOOK are the histogram manager HepHbookManager described in 4.6. Methods which simplify calls to HBOOK and ROOT are described here and ROOT usage is described in **??**. HBOOK histogram output is directed by the HepHbookManager.

The descriptions here only scratch the surface of the analysis tools available in HepTuple and no attempt is made to fully document the HepTuple package[1].

A full example of an analysis program including histogram and ntuple manipulations is in Appendix D.

In this chapter, the following classes are defined for creating histogram objects: HepHist1D, HepHist2D, HepNtuple, and HepHistProf. Each of these objects is defined in its own header file which must be included when referring to the object.

5.1 Booking and Filling Histograms/Ntuples

In AC++ histograms are objects with certain properties (eg. they can be filled). They are referred to by pointers to the objects instead of by numerical *ID*. Therefore the arguments of the methods for booking histograms are very simliar but not quite the same as the FORTRAN HBOOK calls in that the *ID* is the last argument, not the first. The reason for this is that the *ID* is optional since, after booking, the histogram is referred to for filling through the pointer and not through the *ID* . The *ID* is used in PAW or ROOT and can be left to be assigned automatically. Each analysis module gets its own histogram *ID* 's, assigned by the order of booking. If one leaves out the *ID* , then PAW kumacs or ROOT CINT files may no longer work properly when a new histogram is inserted in the booking sequence.

[1] `http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/HepTuple/doc/html/0HepTuple.html`

5.1.1 Book a 1−dimensional histogram

```
_my1dHisto01 = &manager->hist1D(title, nX, xMin, xMax, vmx, id);
```

The method for booking is a member function of the HepFileManager class. The manager (object) in this case was obtained from the fileManager() function described in the prerequisites below has been called and the pointer to the HepFileManager (object) stored in manager.

The result is a pointer to a 1 dimensional histogram object.

This should be done in the beginJob method described in 3.4.

The *id* is the last argument since it is optional and only used to identify the histogram in external programs such as PAW. Within the program, filling is done by referring to the pointer to the histgram returned by the booking method. Normally the pointer is given a name that makes the usage transparent. For example, a pointer to a histogram called "_rapidityHisto" makes it clear that rapidity information should be found in the histogram object to which it points.

Input arguments:

title	histogram title
nX	number of bins
xMin	lower edge of lowest bin
xMax	upper edge of highest bin
vmx	normally set equal to 0.– see HBOOK manual[2] for details.
id	histogram *id* number – nonzero integer

Return Value:

HepHist1D* _my1dHisto01 pointer to the histogram (a HepHist1D histogram object). This pointer should be a private member of the AC++ module. Hence it is declared in the header file[3]. There is one such pointer for *each* histogram and is used to refer to the histogram in the filling method.

Header File: HepTuple/HepHist1D.h

Prerequisites:

manager A pointer to the HepFileManager object, manager, has to obtained from the framework before the histogram can be booked. This is done using the fileManager() method as follows:

[2] `http://wwwinfo.cern.ch/asdoc/hbook_html3/hboomain.html`

[3] Ideally it should be initialized to zero the module class constructor, and receives a value only when this booking occurs.

```
            HepFileManager* manager = fileManager( );
```

_my1dHisto01 This must declared in the header file of the module class as follows:

```
            private:

             HepHist1D* _my1dHisto01;
```

hist1D always deletes an existing histogram and creates a new one.

5.1.2 Book a 2−dimensional histogram

$\boxed{\text{\_my2dHisto01 = \& mananger->hist2D(title, nX, xMin, xMax, nY, yMin, yMax, vmx, id);}}$

hist2D includes the same features as hist1D.

Input arguments:

title	histogram title
nX	number of bins in X
xMin	lower edge of lowest X bin
xMax	upper edge of highest X bin
nY	number of bins in Y
yMin	lower edge of lowest Y bin
yMax	upper edge of highest Y bin
vmx	normally set equal to 0.– see HBOOK manual[4] for details.
id	histogram *id* number – nonzero integer

Return Value:

HepHist2D* _my2dHisto01 pointer to the histogram (a HepHist2D histogram object). This pointer should be a private member of the AC++ module. Hence it is declared in the header file[5]. There is one such pointer for *each* histogram and is used to refer to the histogram in the filling method.

Header File: HepTuple/HepHist2D.h

Prerequisites:

[4] `http://wwwinfo.cern.ch/asdoc/hbook_html3/hboomain.html`

[5]Ideally it should be initialized to zero the module class constructor, and receives a value only when this booking occurs.

manager A pointer to the HepFileManager object, manager, has to obtained from the framework before the histogram can be booked. This is done using the fileManager() method as follows:

```
HepFileManager* manager = fileManager( );
```

_my2dHisto01 This must declared in the header file of the class as follows:

```
private:

  HepHist2D* _my2dHisto01;
```

5.1.3 Book a Profile histogram

$\boxed{\text{\_myProfHist = \&manager->histProf(title, nX, xMin, xMax, yMin, yMax, ChOpt,id);}}$

The profile histogram follows the same rules as a 1d or 2d histogram as described in 5.1.1 and 5.1.2.

5.1.4 Book an Ntuple

$\boxed{\text{\_myNtuple = \&manager->ntuple(title, id);}}$

The *id* at the end is optional but is useful for referring to the histogram in PAW or other such programs.

Input arguments:

title Ntuple title

id Ntuple *id* number – nonzero integer

Return Value:

HepNtuple* _myNtuple pointer to the histogram (a HepNtuple histogram object) This pointer should be a private member of the AC++ module. Hence it is declared in the header file[6]. There is one such pointer for *each* Ntuple.

Header File: HepTuple/HepHistProf.h

Prerequisites:

manager A pointer to the HepFileManager object, manager, has to obtained from the framework before the histogram can be booked. This is done using the fileManager() method as follows:

[6]Ideally it should be initialized to zero the module class constructor, and receives a value only when this booking occurs.

```
HepFileManager* manager = fileManager( );
```

_myNtuple This must declared in the header file of the module class as follows:

```
private:

 HepNtuple* _myNtuple;
```

After this is done

1. The Columns of the Ntuple must be defined as described in 5.1.5.
2. The Ntuple should be cleared as described in 5.1.6

Existing Ntuples will not be overwritten (cf. histograms 5.1.1).

5.1.5 Define the Columns of an Ntuple

$\boxed{\text{\_myNtuple->column(tag,value,default);}}$

Input arguments:

tag	A string indicating the name of the column.
value	Can be used to fill a value immediately, but if used in the beginJob method, it is best to declare this to zero.
default	This can be used to give a value when filling if no argument for filling is given. Filling is described in 5.1.8 and 5.1.9.

Return Value:

NONE

Header File: HepTuple/HepHistNtuple.h

Prerequisites:

_myNtuple a pointer to a HepNtuple must have been defined, the Ntuple booked and all the prerequisites of Ntuple booking must have been followed as described in 5.1.4.

The Ntuple should be cleared as described in 5.1.6.

5.1.6 Clear the Ntuple

```
_myNtuple->clearData();
```

Input arguments:

NONE

Return Value:

NONE

Header File: HepTuple/HepHistNtuple.h

Prerequisites:

_myNtuple	a pointer to a HepNtuple must have been defined, the Ntuple booked, all the prerequisites of Ntuple booking must have been followed as described in 5.1.4, the columns should also have been defined as described in 5.1.5.

5.1.7 Filling a Histogram: Accumulate Step

```
_myHisto->accumulate(valueX,valueY,weight);
```

Input arguments:

valueX	X value to be filled into the histogram.
valueY	Y value to be filled into the histogram.
weight	Weight to be filled into the histogram.

Return Value:

NONE	For a 1d histogram the Y value is ignored. The weight defaults to 1.0. All values are float.
Header File:	HepTuple/HepHist1d.h, HepTuple/HepHist2d.h, or HepTuple/HepHistProf.h, depending on which type of object is being filled.

Prerequisite:

_myHisto

A pointer to a histogram must have been defined through booking as described in 5.1.1,

5.1.8 Filling an Ntuple: Capture Step

$\boxed{\text{\_myNtuple->capture(tag,value);}}$

Input arguments:

tag A string indicating the name of the column as described in 5.1.5.

value Value to be filled into the ntuple.

Return Value:

NONE

Header File: HepTuple/HepHistNtuple.h

Prerequisites:

_myNtuple
1. A pointer to a HepNtuple must have been defined as described in 5.1.4,
2. The columns defined 5.1.5,
3. The ntuple cleared before initial filling as described in 5.1.6.

After the ntuple information is captured it must

1. Be stored as described in 5.1.9[7].
2. Be cleared as was described in 5.1.6.

5.1.9 Filling an Ntuple: Store Step

$\boxed{\text{\_myNtuple->storeCapturedData();}}$

Input arguments:

NONE

Return Value:

NONE

Header File: HepTuple/HepHistNtuple.h

Prerequisites:

_myNtuple
1. A pointer to a HepNtuple must have been defined as described in 5.1.4,

[7]The capture and store steps are equivalent to the HBOOK HFN call in FORTRAN.

2. The columns defined as described in 5.1.5,
3. The ntuple cleared before initial filling as described in 5.1.6.
4. The information should have been captured as described in 5.1.8.

When the information is stored, if the value for a column is not specified, its default as declared in *column* as described in 5.1.5 is used[8].

After storage, the ntuple must be cleared with *clearData* as described in 5.1.6

5.1.10 Sample AC++ program to book and fill histogram, Ntuple

This example is a portion of that in ExampleMyModule/ExampleTrackAnalysis. The full text is included in D The first section must be in the header file and the second in the C++ source code file.

From header file: ExampleTrackAnalysis.hh

```
// Description:
//      Class ExampleTrackAnalysis. This is a simple example of a user module.
//      It books a few histograms, fills them.  Also makes use of talk-to,
//      filtering, and uses a modern (i.e. non-banks) analysis-level
//      data structure in the event record.
// Author List:
//      Ken Bloom
//
//------------------------------------------------------------------------

#ifndef EXAMPLETRACKANA_HH
#define EXAMPLETRACKANA_HH

//----------------------
// Base Class Headers --
//----------------------
#include "FrameMods/HepHistModule.hh"
#ifdef CDF
#include "BaBar/Cdf.hh"
#endif
//-------------------------------------
// Collaborating Class Declarations --
//-------------------------------------
#include "Framework/AbsParmDouble.hh"
class HepHist1D;
class HepNtuple;
```

[8]The capture and store steps are equivalent to the HBOOK HFN call in FORTRAN.

```
//              ---------------------
//              -- Class Interface --
//              ---------------------

class ExampleTrackAnalysis : public HepHistModule {

//--------------------
// Instance Members --
//--------------------

public:

  // Constructors
  ExampleTrackAnalysis( const char* const theName = "ExampleTrackAnalysis",
            const char* const theDescription = "Example user analysis");

  // Destructor
  virtual ~ExampleTrackAnalysis( );
  // Operations
  virtual AppResult      beginJob( AbsEvent* aJob );
  virtual AppResult      event( AbsEvent* event );



private:

  HepHist1D*       _ptHisto;
};

#endif
```

The C++ file: ExampleTrackAnalysis.cc

```
//------------------------------------------------------------------------
// File and Version Information:
//      $Id: ch5.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $
//
// Description:
//      Class MyModule. This is a simple example of a user module. It
//      books a few histograms, fills them.
//
//      The "event" entry point is where you should add code to
//      process event data; define histograms & ntuples in "beginJob"
//
// Author List:
```

```
//      Ken Bloom
//
//------------------------------------------------------------------------

//-----------------------
// This Class's Header --
//-----------------------
#include "ExampleMyModule/ExampleTrackAnalysis.hh"

//-------------
// C Headers --
//-------------
#include <assert.h>
#include <math.h>

//---------------
// C++ Headers --
//---------------

//-------------------------------
// Collaborating Class Headers --
//-------------------------------
#include "AbsEnv/AbsEnv.hh"
#include "HepTuple/HepHist1D.h"
#include "HepTuple/HepHBookNtuple.h"
#include "Edm/EventRecord.hh"
#include "Edm/ConstHandle.hh"
#include "TrackingObjects/Storable/CdfTrackView.hh"
#include "TrackingObjects/Tracks/CdfTrack.hh"

//-----------------------------------------------------------------------
// Local Macros, Typedefs, Structures, Unions and Forward Declarations --
//-----------------------------------------------------------------------

static const char rcsid[] = "$Id: ExampleTrackAnalysis.cc,v 1.3 2000/11/03 22:06

44 bloom Exp $";

//----------------
// Constructors --
//----------------
ExampleTrackAnalysis::ExampleTrackAnalysis(
    const char* const theName,
    const char* const theDescription )
    : HepHistModule( theName, theDescription )
{
}

//--------------
// Destructor --
//--------------
ExampleTrackAnalysis::~ExampleTrackAnalysis( )
{
}

//--------------
// Operations --
//--------------
AppResult ExampleTrackAnalysis::beginJob( AbsEvent* aJob )
{
  //First get access to the object that manages histogram memory space.
  HepFileManager* manager = fileManager( );

  //Book an ntuple.
  _ntuple = &manager->ntuple("Tracks",1);
  //Definecolumns
  _ntuple->column("run",(int)0,(int)0);
  _ntuple->column("event",(int)0,(int)0);
  _ntuple->column("trknum",(int)0,(int)0);
  _ntuple->column("pT",(float)0.,(float)0.);
  //Clear and prepare for filling
  _ntuple->clearData();

  //Book histograms.
  _ptHisto = &manager->hist1D("Track pT", 100, 0.0, 20.0,10);

   return AppResult::OK;
}

AppResult ExampleTrackAnalysis::event( AbsEvent* anEvent )
{
  //By default, this event does not pass the filter.
  bool filter_pass = false;

  //Access the "default" set of tracks in the event by making a CdfTrackView.

  CdfTrackView_h hView; // This is the handle for the "view."
  if (CdfTrackView::defTracks(hView) == CdfTrackView::OK) {

    // The view is now filled with the default tracks, so extract contents.
    const CdfTrackView::CollType & tracks = hView->contents();
    // Now loop over the tracks, doing a double-dereference to get at each.
    for (CdfTrackView::const_iterator it = tracks.begin();
         it != tracks.end(); ++it) {
      const CdfTrack & trk = **it;

      //Extract the pt
      float pt = trk.pt();

      //Fill ntuple.
      _ntuple->capture("run",AbsEnv::instance()->runNumber());
      _ntuple->capture("event",AbsEnv::instance()->trigNumber());
      _ntuple->capture("trknum",(int)trk.id().value());
      _ntuple->capture("pT",pt);

      //Store Data
      _ntuple->storeCapturedData();
      //Clear for next event
      _ntuple->clearData();

      //Fill histogram
      _ptHisto->accumulate(pt);
    }
  }
  return AppResult::OK;
}
```

5.2 Histogram output – the TCL file

histfile	Select the Histogram file name to be used in this job. Note that this parameter is setable from any histogramming module however it can not be altered after the first begin command when it will be opened. **Format** *histfile set <fileName>* **Default** HepHist.dat
createHistoFile	Create the histogram file **Format** *createHistoFile set <true-or.-false>* This is boolean and can be set to "t" or "f". **Default** t
reclen	Select the Histogram file record length. The minimum value is 512 and the max is 65536. **Format** *reclen set <len>* **Default** 1024

Appendix A

Useful References

AC++ Manual	`http://www-cdf.fnal.gov/upgrades/computing/projects/framework/frame_over/frame_over.html`
Coding Guidelines	`http://cdfsga.fnal.gov/computing/coding_guidelines/`
Database Browser Guide	`http://www-cdf.fnal.gov/internal/upgrades/computing/database/browser/browserguide.htm`
Database Browser URL	`http://cdfdbb.fnal.gov:8520/cdfr2/databases`
Data File Catalog Guide	`http://rutpc7.fnal.gov/ratnikov/Docs/DHIOModuleReference.htm`
Event Data Model (EDM) guide	`http://www-cdf.fnal.gov/upgrades/computing/projects/edm/edm.html`
Hbook Manual (URL)	`http://wwwinfo.cern.ch/asdoc/hbook_html3/hboomain.html`
HepTuple Manual (URL)	`http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/HepTuple/doc/html/0HepTuple.html`
TryBos Manual (URL)	`http://www-cdf.fnal.gov/upgrades/computing/projects/trybos/trybos.html`

Appendix B

"Getting Started" Failures

In this Appendix, a few possible failures with getting started are noted and solutions are given. Section B.1 gives hints on what to do if gmake fails. Section B.2 gives hints on what to do if you have trouble with PAW. Throughout this Appendix, commands to be entered are indicated by preceeding it with the prompt `<fcdfsgi2>`.

B.1 *gmake* Trouble

When gmake fails, the first thing to check is if you have run out of disk space. The command to use is:

```
<fcdfsgi2> quota -v
```

with the result

```
Disk quotas for stdenis (uid 8906):
Filesystem    usage   quota   limit    timeleft  files   quota   limit    timeleft
/var/mail         0   51200   51200                  0       0       0
/cdf/scratch  64396 1048576 1048576                705   20480   20480
/cdf/spool        0  512000  512000                  1   10240   10240
/cdf/home     29100  204800  204800               1557   10240   10240
```

In this case all is well because the *Filesystem* called **/cdf/home** has a *usage* of 29100 that is under its quota of 204800 for the space. The number of files, indicated by *files*, is 1557, well under the *quota* of 10240. If you have run out of space and even after clearing out all your files, then you must try the following trick from SRT:

- make a file called *.srtrc* with the following:

  ```
  "$extra_dirs tmp>/cdf/scratch/$USER/releases/$release/tmp
  bin>/cdf/scratch/$USER/releases/$release/bin
  lib>/cdf/scratch/$USER/releases/$release/lib
  results>/cdf/scratch/$USER/releases/results"
  ```

This should all be in a single line in the file *.srtrc*. This will put all your binaries, libraries and results directories from your release on your scratch disk. If you use lots of packages, you will eventually fill the scratch quota on the filesystem /cdf/scratch as well!

- Go to your scratch area and create a directory called *releases*:

  ```
  <fcdfsgi2>cd /cdf/scratch/$USER
  <fcdfsgi2>mkdir releases
  ```

This solution has the problem that you have to remember to clean up both the release your created and what is on your scratch area when you don't want your test release any longer.

B.2 PAW Trouble

If you try to run PAW on fcdfsgi2 as follows:

```
<fcdfsgi2> paw
```

and you get the error:

```
 Calling 2000 version of paw-X11

 ******************************************************
 *                                                    *
 *            W E L C O M E    to   P A W             *
 *                                                    *
 *       Version 2.11/11       9 November 1999        *
 *                                                    *
 ******************************************************
 Workstation type (?=HELP) <CR>=1 :
 Version 1.26/04 of HIGZ started
 ***** ERROR in IOPWK : Can't open DISPLAY
 ***** ERROR in IACWK : Workstation is not open
 ***** ERROR in ISWKWN : Invalid workstation window parameters
 ***** ERROR in ISWKVP : Invalid workstation window parameters
PAW >
```

you need to fix up your DISPLAY enviroment variable: Suppose your PC is called cdf01.fnal.gov. Then you must type:

```
setenv DISPLAY cdf01.fnal.gov:0.0
```

for c shell or

```
DISPLAY=cdf01.fnal.gov:0.0; export DISPLAY
```

in bash. You can also use:

```
setenv DISPLAY ‘echo $REMOTEHOST‘:0.0
```

or

```
DISPLAY=‘echo $REMOTEHOST‘:0.0 ; export DISPLAY
```

This works on any PC and is transferable.

By the way, it is a lot easier to check these connections if you dont use PAW, but XLOGO. The command

```
<fcdfsgi2> xlogo
```

will display an “X” in window on your PC. If this fails, the command and error message is:

```
<fcdfsgi2> xlogo
Error: Can't open display:
```

Another mode of failure is

```
<fcdfsgi2> paw

 Calling 2000 version of paw-X11

 ******************************************************
 *                                                    *
 *            W E L C O M E    to   P A W             *
 *                                                    *
 *       Version 2.11/11       9 November 1999        *
 *                                                    *
 ******************************************************
 Workstation type (?=HELP) <CR>=1 :
 Version 1.26/04 of HIGZ started
Xlib: connection to "cdf01.fnal.gov:0.0" refused by server
Xlib: Client is not authorized to connect to Server
 ***** ERROR in IOPWK : Can't open DISPLAY
 ***** ERROR in IACWK : Workstation is not open
 ***** ERROR in ISWKWN : Invalid workstation window parameters
 ***** ERROR in ISWKVP : Invalid workstation window parameters

PAW >
```

The difference from the problem shown above is the Xlib error messages. This tells you that the problem is that you not allowed windows to be popped upon your PC.

To overcome this failure type

```
xhost + fcdfsgi2.fnal.gov
```

in some window on your LINUX PC.

Again, it is easier to diagnose this error with XLOGO. When the error occurs, you find the session gives:

```
fcdfsgi2> xlogo
Xlib: connection to "cdf01.fnal.gov:0.0" refused by server
Xlib: Client is not authorized to connect to Server
Error: Can't open display: cdf01.fnal.gov:0.0
```

Appendix C

Program Structure

```
    main                   main program
    |
    +-------|           PROGRAM INITIALIZATION
    |       |
    |       |
    |       +--  a->beginJob         initialization for Module a
    |       +--  b->beginJob         initialization for Module b
    |       .
    |       +--  n->beginJob         initialization for Module n
    |       +--  myModule->beginJob  user initialization    <---
    |
+->-+--DHInput->event           READ EVENTS
|       |              Take action based on event type, file status
|   -----------
|   |     | | |
|   |     | | |   open input files; queue tape requests; read next record
|   |     | | |
|   |     | | |   terminate job if eof or time limit or ...
|   |     | | |
|   |     | | +-- a->endJob   event analysis from Module a
|   |     | | +-- b->endJob   event analysis from Module b
|   |     | | .
|   |     | | +-- n->endJob   event analysis from Module N
|   |     | | +-- myModule->endJob
|   |     | |
|   |     | +-- a->beginRun   event analysis from Module a
|   |     | +-- b->beginRun   event analysis from Module b
|   |     | .
|   |     | +-- n->beginRun   event analysis from Module N
|   |     | +-...myModule-> beginRun called for every new run
|   |     | |
|   |---<---|
|   |     |   close input files; compute stats, reset histos
|   |     |
|   |     +-- a->endRun   event analysis from Module a
```

```
 |   |     +-- b->endRun   event analysis from Module b
 |   |     .
 |   |     +-- n->endRun   event analysis from Module N
 |   |     +-...myModule-> endRun called for every new run
 |------<--|
 |   |
 |   +----      PROCESS ONE EVENT
 |   | | |
 |   | | +-- a->event   event analysis from Module a
 |   | | +-- b->event   event analysis from Module b
 |   | | .
 |   | | +-- n->event   event analysis from Module N
 |   | | |
 +-<-+ | +-- myModule->event  user event analysis         <---
       |
       |
       +-- a->other   other analysis from Module a
       +-- b->other   other analysis from Module b
       .
       +-- n->other   other analysis from Module N
       |
       +-- myModule->other  user event analysis
```

```
Arrows (<---) indicate the important user routines.
```

Appendix D

Full Example of an Analysis Module

This is an full example of an analysis module. The first file is the header file for the module, containing the declarations of the methods, the parameters for the module and any other private members or classes needed by the module. The second file is the C++ code for the module. The third file is the build file for the module.

These files must be placed into the correct directory structure for the build to occur properly. This is currently done by checking out the example from the CVS repository as decribed in Ch. 2. In addition, dependencies of the module on other objects in the CDF offline needs to be specified in another file. This is described below.

D.1 Description of the Directory Structure

Software Release Tools are used to create directory structures into which the code for the analysis modules are placed. This section first describes the *head* of the test release, then goes on to describe the contents of the individual package, in this case, a package corresponding to the ExampleMyModule.

D.1.1 The Head of the Release

After one has followed the instructions for creating a "test release" as described in Ch. 2, one has in the subdirectory *ana* the following contents:

```
GNUmakefile  doc          lib          results      ups
bin          include      man          tmp
```

Reference to all files then is made relative to this directory, referred to as the *head* of the test release. The only file in this directory is the GNUmakefile and one never edits this file. The *doc, lib, man, tmp,* and *include* areas point to the libraries, man pages, temporary and include files of any additional software that is added to this structure. The .cc, .hh and build files for a module are examples of software to be added. This is done in the context of a package. The easist way to proceed is to check out an example package and modify the .cc, .hh and build files as described in the next section.

The *bin* area contains all binary executables that are built by the makefiles.

D.1.2 Packages

After the `addpkg -h ExampleMyModule` command has been issued, the *ana* directory at the head of the release contains:

```
ExampleMyModule  doc              man              ups
GNUmakefile      include          results
bin              lib              tmp
```

The interesting directories are those referring to the *package* ExampleMyModule. This is where the .cc, .hh and files for building the executable are found.

The ExampleMyModule directory contains the following:

```
BuildExampleMyModule_test.cc  MyModule.hh
CVS                           README
ExampleTrackAnalysis.cc       link_ExampleMyModule.mk
ExampleTrackAnalysis.hh       run.tcl
GNUmakefile                   run_track.tcl
MyModule.cc
```

The file BuildExampleMyModule_test.cc is the *build* file and specifies the various modules that will comprise the binary executable, the AC++ program. It is shown in D.4.

There are two files with *.cc* and *.hh* extensions corresponding to two modules, one called ExampleTrackAnalysis and the other ExampleMyModule. The modules are the classes whose methods are described in Ch. 3. The ExampleMyModule will be described in the following sections.

D.2 The Header File

The name "MyModule" is defined by the Build (see D.4) and and Makefile (see D.5) files. This name can be changed but must be done so consistently throughout.

The header file: MyModule.hh

```
//--------------------------------------------------------------------
```

```
// File and Version Information:
//  $Id: app_exampcode.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $
//
// Description:
//  Class MyModule. This is a simple example of a user module. It
//  books a few histograms, fills them.
//
// Environment:
//  Software developed for CDF.
//
// Author List:
//  Liz Sexton-Kennedy
//
//------------------------------------------------------------------------

#ifndef MYMODULE_HH
#define MYMODULE_HH

//----------------------
// Base Class Headers --
//----------------------
#include "FrameMods/HepHistModule.hh"

//-------------------------------------
// Collaborating Class Declarations --
//-------------------------------------
class HepHist1D;
//      ---------------------
//      -- Class Interface --
//      ---------------------

class MyModule : public HepHistModule {

//--------------------
// Instance Members --
//--------------------

   public:

      // Constructors
      MyModule( const char* const theName = "MyModule",
        const char* const theDescription = "Example user analysis");

      // Destructor
      virtual ~MyModule( );
      // Operations
      virtual AppResult      beginJob( AbsEvent* aJob );
      virtual AppResult      event( AbsEvent* event );
      virtual AppModule*     clone( const char* cloneName );

      const char*  rcsId( ) const;

   private:

      HepHist1D*       _EHisto;
      HepHist1D*       _MassHisto;
};

#endif
```

The elements of this file are as follows:

C++ Macro Declarations The declarations

```
#ifndef MYMODULE_HH
#define MYMODULE_HH
```

are at the start of the file, after comments. These are important because it allows the header file to be imbedded in another header file. Since one is never quite aware of what is in a header file without explicitly looking at the code, if a header secretly includes another header file, then having a second inclusion of the same header file will cause no harm if these statements are included and the

```
#endif
```

statement is included at the very end.

Base Class Include This being a histogramming module, requires that the header for the histogramming module be included. All AC++ modules inherit from a base class. For a detailed explanation of the various optional classes, see **??**. For most purposes, the histogramming base class will do.

Collaborating Classes The histogramming classes must be declared in the header so that the pointers may be declared as private members of the AC++ module class.

class MyModule : public HepHistModule { This is the declaration of the uers's module itself, inheriting from a histogramming module class, HepHistModule. For a detailed explanation of the various optional classes, see **??**. For most purposes, the histogramming base class will do. This is named HepHistModule. This and associated files have been connected in by the "setup cdfsoft2 3.11.0" command.

public: The public members of the AC++ module class follow.

Constructor The constructor is declared with the module name and the description of the module as was described in 3.8.1.

Destructor The destructor is declared with the module name and the description of the module as was described in 3.8.2.

User Methods The user methods are declared. Here the beginJob(), event() and clone() methods described in 3.4, 3.5, and 3.7.5 are declared.

RCS ID The revision control system identifier is added. This is to allow one to query as to the CVS revision number so that backwards compatibility may be maintained. When releasing code for public use, backwards compatibility becomes a serious issue and use of this is important. Further information on the usage of this may be found in **??**.

private: The private member declarations follow. This will include talk-to parameters and histogram pointers. In this case there are two histograms to be booked and hence the two pointers are required. Because of the usage of the histogram class names, it was necessary to have already declared that there exists this histogram class.

D.3 The C++ File

The C++ file: MyModule.cc

```
//------------------------------------------------------------------------
// File and Version Information:
//  $Id: app_exampcode.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $
//
// Description:
//  Class MyModule. This is a simple example of a user module. It
//  books a few histograms, fills them.
//
//      The "fillHistograms" entry point is where you should add code to
//      process event data; define histograms & ntuples in "beginJob"
//
// Environment:
```

```
//  Software developed for CDF.
//
// Author List:
//      Liz Sexton-Kennedy
//
//---------------------------------------------------------------------------
//#include "ISOcxx/ISOcxx.h"
//----------------------
// This Class's Header --
//----------------------
#include "ExampleMyModule/MyModule.hh"

//-------------
// C Headers --
//-------------
#include <cassert>

//---------------
// C++ Headers --
//---------------

//-------------------------------
// Collaborating Class Headers --
//-------------------------------
#include "HepTuple/HepHist1D.h"

#include "Edm/EventRecord.hh"
#include "Edm/ConstHandle.hh"
#include "StorableBanks/CMUO_StorableBank.hh"

//-----------------------------------------------------------------------
// Local Macros, Typedefs, Structures, Unions and Forward Declarations --
//-----------------------------------------------------------------------

static const char rcsid[] = "$Id: app_exampcode.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $";

//----------------
// Constructors --
//----------------
MyModule::MyModule(
    const char* const theName,
    const char* const theDescription )
    : HepHistModule( theName, theDescription )
{
}
//--------------
```

```
// Destructor --
//--------------
MyModule::~MyModule( )
{
}

//--------------
// Operations --
//--------------
AppResult MyModule::beginJob( AbsEvent* aJob )
{
  HepFileManager* manager = fileManager( );
  assert( 0 != manager );
  _EHisto = &manager->hist1D( "Muon Energy", 100, .5, 50.0, 100 );
  assert( 0 != _EHisto );
  _MassHisto = &manager->hist1D( "Pair Mass", 100 , 0.0, 5.0, 101 );
  assert( 0 != _MassHisto );
   return AppResult::OK;
}

AppResult MyModule::event( AbsEvent* anEvent )
{
   const double jpsi_mass       = 3.09688 ;
   const double mass_window_halfwidth = 3.0000 ;

/*==========================================================================*\
 * Loop over distinct CMUO bank pairs within the event
\*==========================================================================*/

   const double min_muon_pt     = 1.5;  // past the steel

   for (EventRecord::ConstIterator muon_iter1(anEvent, "CMUO_StorableBank") ;
    muon_iter1.is_valid() ; ++muon_iter1)
   {
      for (EventRecord::ConstIterator muon_iter2(muon_iter1.peek_ahead()) ;
       muon_iter2.is_valid() ; ++muon_iter2)
      {
     ConstHandle<CMUO_StorableBank> h_CMUO_1(muon_iter1) ;
     ConstHandle<CMUO_StorableBank> h_CMUO_2(muon_iter2) ;

//--------------------------------------------------------------------------
// Cut on Muon Pt and Charge
//--------------------------------------------------------------------------
     if ( (h_CMUO_1->pt() > min_muon_pt) &&
          (h_CMUO_2->pt() > min_muon_pt) &&
          (h_CMUO_1->em_charge() + h_CMUO_2->em_charge() == 0) )
     {

//------------------------------------------------------------------------------
// Determine the pair mass of JPsi Candidate
//------------------------------------------------------------------------------
        double px     = h_CMUO_1->px()     + h_CMUO_2->px() ;
        double py     = h_CMUO_1->py()     + h_CMUO_2->py() ;
        double pz     = h_CMUO_1->pz()     + h_CMUO_2->pz() ;
        double energy = h_CMUO_1->energy() + h_CMUO_2->energy() ;

        double pair_mass2 = energy*energy - px*px - py*py - pz*pz ;
        double pair_mass  = 0.0 ;

        if (pair_mass2 >= 0.0)
        {
           pair_mass = sqrt(pair_mass2) ;
        }
        else
        {
           pair_mass = -sqrt(-pair_mass2) ;
        }

//------------------------------------------------------------------------------
// Cut on JPsi Candidate mass
//------------------------------------------------------------------------------
        if ( fabs(pair_mass - jpsi_mass) < mass_window_halfwidth )
        {
           energy = h_CMUO_1->energy();

//------------------------------------------------------------------------------*\
// JPsi Candidate found now histogram fill it
//------------------------------------------------------------------------------*/
           _EHisto->accumulate( energy );
           _MassHisto->accumulate( pair_mass );
        }
     }
      }

   }

   return AppResult::OK;
}

AppModule*
MyModule::clone(const char* cloneName)
{
  return new MyModule(cloneName,"this module is a clone MyModule");
}

const char *
MyModule::rcsId( ) const
{
   return rcsid;
}
```

D.4 The Build File

```
//-----------------------------------------------------------------------------
// File and Version Information:
//  $Id: app_exampcode.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $
//
// Description:
//  Class AppUserBuild. This class must be provided by the user of
//  the framework in order to build an application. It must define
//  the modules that are to form the basis of the application.
//
// Environment:
//  Software developed for the CDF Detector
//
//---------------------------------------------------------------------------

//-----------------------
// This Class's Header --
//-----------------------
#include "Framework/APPUserBuild.hh"

//-------------------------------
// Collaborating Class Headers --
//-------------------------------
#include "ExampleMyModule/MyModule.hh"
#include "ExampleMyModule/ExampleTrackAnalysis.hh"
#include "FrameMods/addCDFrequiredModules.hh"
#include "FrameMods/addAllStorableObjects.hh"
#include "FrameMods/hbook/HepHbookManager.hh"
//#include "FrameMods/root/HepRootManager.hh"

//---------------------------------------------------------------------
// Local Macros, Typedefs, Structures, Unions and Forward Declarations --
//---------------------------------------------------------------------
```

```
static const char rcsid[] = "$Id: app_exampcode.tex,v 1.2 2001/01/15 13:53:45 stdenis Exp $";

//----------------
// Constructors --
//----------------

AppUserBuild::AppUserBuild( AppFramework* theFramework )
    : AppBuild( theFramework )
{

    addCDFrequiredModules( this );

    // This is needed because the ExampleTrackAnalysis module wants to
    // look at the output of ProductionExe
    addAllStorableObjects( );

    AppModule* aModule;

    // This is a utility module whose only purpose is to select a histogram
    // manager type:
    // aModule = new HepRootManager( );
    // comment out the next line and uncomment the above line if you want
    // root instead.
    aModule = new HepHbookManager( );
    add( aModule );

    aModule = new MyModule( );
    add( aModule );

    aModule = new ExampleTrackAnalysis( );
    add( aModule );

    // Any any other modules you want to link here...
}

//--------------
// Destructor --
//--------------

AppUserBuild::~AppUserBuild( )
{
}
const char * AppUserBuild::rcsId( ) const
{
   return rcsid;
}
```

D.5 The Makefile

```
# example Makefile for CDF packages
#
# uses SoftRelTools/standard.mk
#
############################################################
# file lists (standard names, local contents)

# include file products
INC =

# library product
LIB = libExampleMyModule.a

# library contents
#   Note: source files with main() definitions do not go
#   into the library.
#   Any file copied from AppUserBuild_template.cc should
#   also be skipped.
skip_files := BuildExampleMyModule_test.cc
LIBCCFILES  = $(filter-out $(skip_files), $(wildcard *.cc))
LIBFFILES   = $(wildcard *.F)
LIBCFILES   = $(wildcard *.c)

override LINK_ExampleMyModule += ExampleMyModule
override LINK_FrameMods       += ExampleMyModule
override LINK_TrackingMods    += ExampleMyModule
override LINK_FrameMods_hbook += ExampleMyModule
override LINK_FrameMods_dump  += ExampleMyModule

-include PackageList/link_all.mk


# binary products
BINS = ExampleMyModule_test
COMPLEXBIN = ExampleMyModule_test
BINCCFILES = BuildExampleMyModule_test.cc

###########################################################
include SoftRelTools/arch_spec_STL.mk
include SoftRelTools/standard.mk
```

Appendix E

Advanced Features of the CalibrationManager Required Module

add **Format:** *add <db identifier> <db type> <db name>*

The database identifier is any character string that is then used in the code to open the database for a specific table.

The database is one of the following:

OTL	Oracle Template Library – this is the Oracle Database
Text	The text database. See **??** for details on what a text database is and how to create one.
msql	MSQL database: a freeware database

The database name depends on the database used:

OTL The username, password and name of the database, written in the format:

Format: *<username>/<password>@<database>*

The syntax

Format: */@cdfondev*

can be used by all CDF users who have an account on the online development machine. CDF users are automatically given an account on the Oracle development database so the username and passwords are not needed since Oracle leaves the responsibility for logging in to the operating system.

Text The directory containing the text database. See **??** for details on what a text database is and how to create one.

msql The directory containing the MSQL database.

IomapFile **Format:** *IomapFile set <iomap-filename>*

Default: NONE (takes the one in DBManager/iomap.txt).

Description: The Name of the DBManager IoMap file you wish to use: if no value is entered, the DBManager defaults are used.

The full directory path may be included in the *iomap-filename*. If no path is specified, the present working directory is used. Individual entries in the iomap file name follow the same syntax rules as the *add* command.
When creating an IoMapFile, on must set the access to 600:

```
chmod 600 iomap.txt
```

Version **Format:** *Version set <version>*

Default: 9999999

Description: Enter the Calibration version you wish to use.

LoadAll **Format:** *LoadAll set <true-or-false>*
This is boolean and can be set to "t" or "f".

Default: f

Description: Indicates that all database drivers should be loaded from the default db-id

Jobset **Format:** *Jobset set <jobset-number>*

Default: -1

Description: jobset # (Force the use of this jobset for this job). A jobset is used to relate the valid calibrations to the ones that should be used to analyze the data. This rather byzantine relation is described in **??**

Debug **Format:** *Debug set <true-or-false>*
This is boolean and can be set to "t" or "f".

Default: f

Description: Set this to false to stop any queries to USED_SETS – very dangerous if you want calibration data and are not an expert!

UseKeyDB **Format:** *UseKeyDB set <true-or-false>*
This is boolean and can be set to "t" or "f".

Default: t

Description: Set this to true to turn on warning/error messages if you are using the KEYDB For information on the Key database see **??**.

Appendix F

Process Names

Process names are built in the form $< destination >\_< system >\_< calib\_mode >$. Some process names do not follow this pattern and will eventually stop being used.

The most process names are:

PROCESS_NAME	DEST	SYS	MODE	USAGE
PROD_COMM_CDF	PROD	CDF	COMM	Analysis of all PAD data from the commissioning run

Details on other process names, and the calibration modes, systems and destinations are given in the following pages.

The table below describes the process names and the following tables give the possible destinations, system and calibration modes.

PROCESS_NAME	DESTINATION	SYSTEM	CALIB_MODE
COT_TEST	UNKNOWN	COT	TEST
L3_TRIGGER	L3	TRIG	UNDEFINED
MUON_TEST	UNKNOWN	MUON	TEST
PRODUCTION_TEST	PROD	CDF	TEST
PESCALIB_TEST	UNKNOWN	SMX	TEST
RUN1_CTC_CONSTANTS	UNKNOWN	CTC	RUN1
TEST	UNKNOWN	CDF	UNDEFINED
CALOR_DOWNLOADS	CRATES	CAL	UNDEFINED
CALOR_DOWNLOADS_FLASHER	CRATES	CAL	FLASH
CR_S1_BW4_132	UNKNOWN	SVX	BW4S1
CRATES_PHYSICS_CAL	CRATES	CAL	PHYSICS
CRATES_SOURCE_CAL	CRATES	CAL	SOURCE
CRATES_FLASH_CAL	CRATES	CAL	FLASH
PROD_COMM_COT	PROD	COT	COMM
PROD_TEST_COT	PROD	COT	TEST
PROD_COMM_CAL	PROD	CAL	COMM
PROD_TEST_CAL	PROD	CAL	TEST
PROD_TEST_MUON	PROD	MUON	TEST
PROD_BW2S1_SVX	PROD	SVX	BW2S1
PROD_BW3S1_SVX	PROD	SVX	BW3S1
PROD_BW4S1_SVX	PROD	SVX	BW4S1
PROD_BW4S2_SVX	PROD	SVX	BW4S2
PROD_BW4S3_SVX	PROD	SVX	BW4S3
PROD_BW4S4_SVX	PROD	SVX	BW4S4
PROD_BW5S1_SVX	PROD	SVX	BW5S1
PROD_BW6S1_SVX	PROD	SVX	BW6S1
PROD_BW7S1_SVX	PROD	SVX	BW7S1
PROD_BW7S2_SVX	PROD	SVX	BW7S2
PROD_COMM_CDF	PROD	CDF	COMM
PROD_TEST_CDF	PROD	CDF	TEST
L3_TEST_TRIG	L3	TRIG	TEST
L3_PHYSICS_TRIG	L3	TRIG	PHYSICS

The most recent version of this table may be found on the web [1] in the database browser [2].

[1] `http://cdfdbb.fnal.gov:8520/cdfr2/databases`
[2] `http://www-cdf.fnal.gov/internal/upgrades/computing/database/browser/browserguide.htm`

The destinations are:

DESTINATION	DESCRIPTION
UNKNOWN	Unknown user of this process def
PROD	Reconstruction executable
SIM	Monte Carlo simulation
L1	Level 1 trigger
L2	Level2 trigger processors
L3	Level3 trigger farm
CRATES	Front end crates
TOP	Top analyses
BOTTOM	B physics analyses
QCD	QCD analyses
PAD	General post-reconstruction analyses
CONSUMER	Online consumers

The systems are:

SYSTEM	DESCRIPTION
CTC	Central Tracking Chamber (RUN1)
COT	Central Outer Tracker
CLC	Luminosity Counter
CAL	Calorimeter and Shower Max
SMX	Shower Max detectors
SVX	Silicon detectors
MUON	All muon detectors
TOF	Time of Flight
TRIG	Trigger L1/L2/L3
DAQ	Data Acquisition
FWD	All forward detectors, BSC, etc.
CDF	Composite CDF detector

The valid modes are:

CALIB_MODE	DESCRIPTION
UNDEFINED	Unknown conditions of use
RUN1	Run 1 data
TEST	Testing
PHYSICS	Normal physics running
COMM	Fall 2000 commissioning run
BW2S1	Silicon Bandwidth 2 setting
BW3S1	Silicon Bandwidth 3 setting
BW4S1	Silicon Bandwidth 4 setting
BW4S2	Silicon Bandwidth 4 setting
BW4S3	Silicon Bandwidth 4 setting
BW4S4	Silicon Bandwidth 4 setting
BW5S1	Silicon Bandwidth 5 setting
BW6S1	Silicon Bandwidth 6 setting
BW7S1	Silicon Bandwidth 7 setting
BW7S2	Silicon Bandwidth 7 setting
SOURCE	Calorimeter source runs
FLASH	Calorimeter LED and Xe flasher runs

For each system, only certain modes are valid. The correspondence is shown here:

SYSTEM	CALIB_MODE
TRIG	UNDEFINED COMM TEST PHYSICS
SVX	TEST BW2S1 BW3S1 BW4S1 BW4S2 BW4S3 BW4S4 BW5S1 BW6S1 BW7S1 BW7S2
MUON	TEST COMM PHYSICS
CTC	RUN1
COT	TEST COMM PHYSICS
CAL	UNDEFINED TEST COMM PHYSICS SOURCE FLASH
SMX	TEST PHYSICS SOURCE
TOF	TEST
FWD	TEST
CLC	TEST
CDF	UNDEFINED TEST COMM PHYSICS

Appendix G

Calibration Status List

The table below describes the possible calibration status and its meening. The most recent version of this table may be found on the web [1] in the database browser [2].

DATA_STATUS	DESCRIPTION
COMPLETE	Results contain a complete set of channels as defined in CALIB_PROPERTIES
RAW	Results are unfiltered
GOOD	Results have bad channels filtered out
FILTERED	DEPRECATED: use GOOD instead
TIGHT	Results had unusually tight filtering
RELAXED	Results had unusually relaxed filtering
BAD	Results were labelled bad for later study
TEST	Results from a test run of the DB software
DERIVED	Results were derived from multiple runs but are not necessarily COMPLETE
UNDEFINED	Results are unknown/unusable
ORPHANED	Unusable, results were written without an entry in CALIBRUNLISTS

[1] `http://cdfdbb.fnal.gov:8520/cdfr2/databases`
[2] `http://www-cdf.fnal.gov/internal/upgrades/computing/database/browser/browserguide.htm`

Appendix H

Advanced Parameters in Module Talk-To's

There are some advanced ways to set parameters in the TCL file. These advanced methods include lists H.2,

H.1 ENUM

AbsParmEnum _parname('command-string',this,'enum-string')

The procedure for adding one of these to a module is:

1. Add an AbsParmEnum as a private member of your module in its header:

   ```
   private:
     AbsParmEnum _parname;
   ```

2. intialize it in the constructor of your module:

   ```
    MyModule::MyModule(const char* const name,
                       const char* const description)
      : AppModule(name, description)
      , _parname("cutList",this,"LOOSE")
    {
   ```

   It is possible then to choose the ENUM to work off the integer values or strings representing the enum tags. In the initializer above, the tag has been chosen. If the ENUM value was used, it would be initialized (in this case) to the integer "1".

3. In body of the module constructor defines the ENUM options:

   ```
   list< string > enumNames;
   enumNames.push_back( "LOOSE" );
   enumNames.push_back( "TIGHT" );
    _parnme.initValidList( enumNames );
   ```

4. In the body of the constructor, set the description:

```
_parname.addDescription(  "Select cuts to use. The syntax of the
                           command is \n\ cutList set <choice>,
                           where <choice> is the enum string from
                           above.");
```

5. Add the parameter to a menu:

   ```
   commands( )->append( &_processMenu);
   ```

6. In using the beginJob(), beginRun() or event(), method, access the values:

   ```
     if(_parnmame.value()=="LOOSE"){
        _applyLooseCuts();
     }else
        _applyTightCuts();
     }
   ```

H.2 List of Parameters

Lists of parameters may be accessed as described in this section. The procedure for adding one of these to a module is:

1. Add an AbsParmList as a private member of your module in its header:

   ```
   private:
     AbsParmList<double> _dList; // (double is merely an example)
   ```

2. intialize it in the constructor of your module:

   ```
    MyModule::MyModule(const char* const name,
                       const char* const description)
      : AppModule(name, description)
      , _dList("dList",this,2,5,1.0)
    {
   ```

   This would initialize a list of double dList with an initial size of 2, a maximum size of 5 and each element initialized to 1.0

3. Optionally set the defaults:

   ```
    _dList.setDefault(myList);
   ```

   or

   ```
    _dList.setDefault();
    _dList.addDefault(val1);
    _dList.addDefault(...);
    _dList.addDefault(valN);
   ```

4. In the body of the constructor, set the description:

```
_dList.addDescription("blah");
```

5. Add the parameter to a menu:

```
commands()->append(&_dList);
```

6. In the beginJob() or event() methods access the list with:

```
for (AbsParmList<double>::ConstIterator i=_dList.begin();
     i!=_dList.begin(); ++i;) {
  cout << *i << endl; //
```