Analyzing STAR Data if You Are New to STAR:
a MuDST Tutorial

Based on the excellent tutorials by Dan Magestro and Akio Agowa

by
Jim Thomas
Lawrence Berkeley National Laboratory

First edition written in 2006 but now updated to 2011

please send suggestions and comments to jhthomas@lbl.gov

If you are new to STAR but familiar with Root…

- STAR data analysis techniques use a few distinct styles and conventions that are unique to STAR … but which are very powerful
  - Cons
  - Makers
  - μDSTs
  - The Scheduler

Cons

- STAR uses ‘cons’ to compile, link, and make files
  - You should create a sub-directory called ‘StRoot’ in your primary work area
    - Create a sub-directory called ‘HelloWorld’ under ‘StRoot’
      - Put ‘HelloWorld.cxx’ and ‘HelloWorld.h’ into ‘StRoot/HelloWorld/’
  - Now type ‘cons’ from your primary directory and your files will be automatically compiled and linked
    - No Make files! Somebody else does all that work for you …
    - This works even if your files refer to other STAR or ROOT software

- You can change the version of root, the compilers, and cons
  - The command ‘starpro’ issued at the command line will give you the production version of all STAR software
  - Similarly, ‘starold’, ‘starnew’, and ‘stardev’ give the newest stable version of the software or the developmental version of the software.
  - A specific version of older star software can be achieved with the ‘starver’ command (ie. ‘starver SL09b’)

Check out your own copy of STAR Software

- You can checkout your own copy of STAR software, change it, cons it, and 'root4star' will use the local copy in your StRoot area before using the global STAR copy of that library
  - From your primary directory, execute the following command
    - > cvs co StRoot/StMuDSTMaker

    (p.s. note capitalization of DST)
  - This will create the StRoot sub-directory and fill it with all of the StMuDstMaker routines. You can now compile the software
    - > stardev
    - > cons

    cvs contains the latest software that may only compile in dev
  - The local copy of the files (and your changes) in StRoot/StMuDSTMaker will be the default until you delete the directory.
    - You may not want this result … so check out software wisely

- STAR offline software is documented in CVS & D'Oxygen

  CVS: http://www.star.bnl.gov/cgi-bin/protected/cvsweb.cgi/StRoot/

  D'Ox: http://www.star.bnl.gov/webdata/dox/html/classes.html

  These links can be found from homepage => computing => CVS Tools

  or homepage => computing => Offline doc => Classes => Class Index

Makers

- STAR analysis software uses ‘Makers’
  - For our purposes, this is a recommended style for structuring your event analysis programs
  - A Maker has three required member functions
    - Init()
    - Make()
    - Finish()
  - You can do elegant things with Makers, including chaining several of them together, but for now we will introduce the style

- The elements of the “HelloWorldMaker.cxx”

```
Int_t HelloWorldMaker::Init( )
  { // Do once at the start of the analysis, create histograms, etc. }

Int_t HellowWorldMaker::Make( )
  { // Do every event
    cout << "Hello World" << end ;
    return kStOK ;   }

Int_t HellowWorldMaker::Finish( )
  { // Do once at the end of the analysis, close files, etc. }
```

Execute the Maker from a macro like this …

```
void HelloWorld( )
{
  // Load libraries
  gROOT   -> Macro("loadMuDst.C")
  gSystem -> Load("HelloWorldMaker.so") ;

  // List of member links in the chain
  StChain* chain = new StChain ;
  HelloWorldMaker* Hello = new HelloWorldMaker( ) ;

  Int_t nEvents = 10 ;

  // Loop over the links in the chain
  chain -> Init() ;
  chain -> EventLoop(1,nEvents) ;
  chain -> Finish() ;

  // Cleanup
  delete chain ;
}
```

You can make a list of several makers

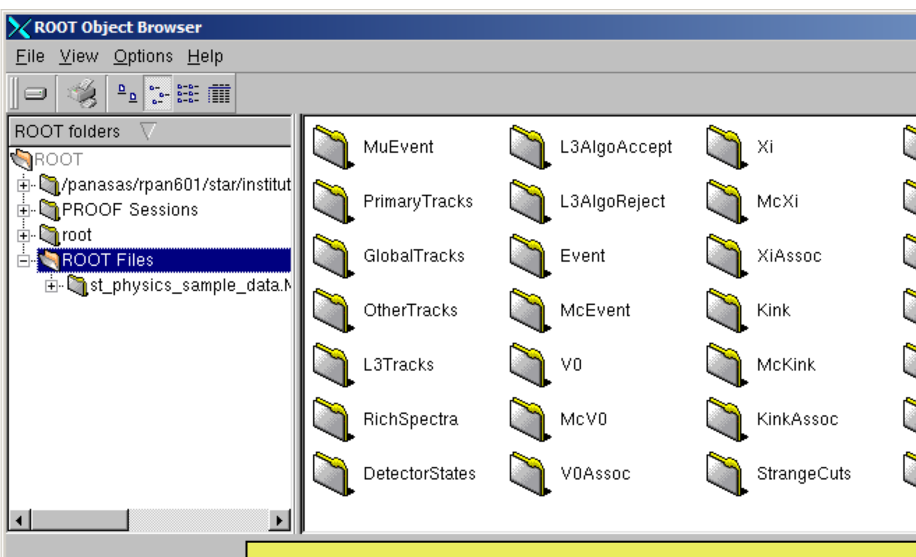
and they will be executed, in order, here.

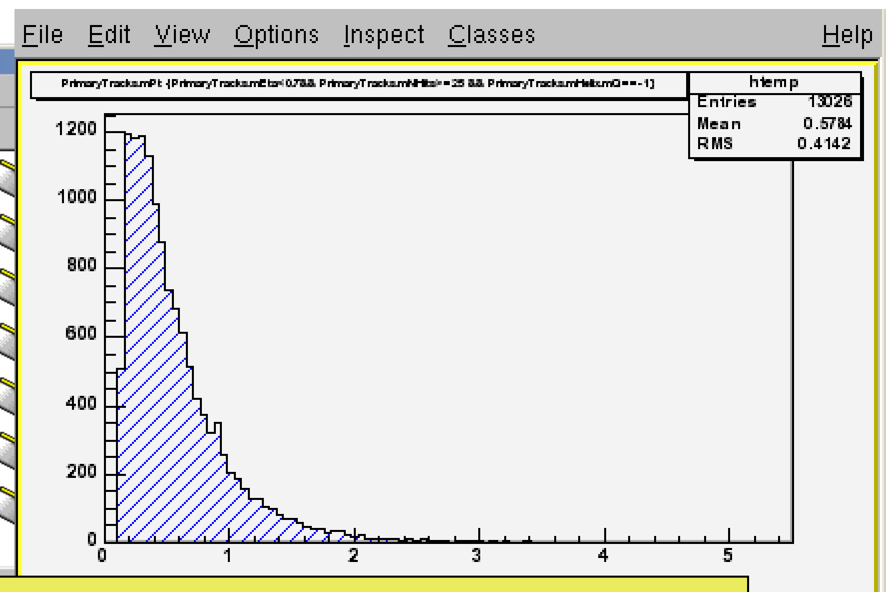
Full example at rcf: **`/star/institutions/lbl/jhthomas/muDstTutorial/HellowWorldExample`**

How Do I Read a STAR Data Tape?

- One way is to open a MuDst file, a TBrowser, and off you go
  - Fast way to read the data (just requested quantities are read)
  - Good for quick checks, data sanity, etc.
  - TChains of TFiles can be used to increase stats (not shown here)

```
root[0] TFile f("st_physics_sample_data.MuDst.root")
root[1] TBrowser b
root[2] MuDst->Draw("PrimaryTracks.mPt","PrimaryTracks.mEta<0.7 &&
        PrimaryTracks.mNHits>=25 && PrimaryTracks.mHelix.mQ==-1")
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```





Sample MuDst data file at rcf: `/star/institutions/lbl/jhthomas/muDstTutorial/`

A Better Way to Read a MuDst

- Let StMuDstMaker do the work for you
  - StMuDstMaker handles all IO
  - All event and track information is accessed with simple class methods
  - Read only the data you want without wasting time … no need for picoDSTs
    - Example: you can read the primary tracks and skip over the global tracks without paying an IO penalty
  - Essential for physics analysis with multiple primary vertices (≥ 2005)

```
StMuDstMaker *muDstMaker = new StMuDstMaker( int mode,
                                             int nameMode,
                                             const char *dirName="./",
                                             const char *fileName="",
                                             const char *filter=".",
                                             int maxfiles=10,
                                             const char *name="MuDst" )
```

Commonly used branches, methods

STAR

GlobalTracks, PrimaryTracks

```
StMuTrack *muTrack = (StMuTrack*) GetTracks.Next();
```

track quantities

```
muTrack->eta()              // pseudorapidity
muTrack->phi()              // az.angle (radians)
muTrack->pt()               // transverse mom.
muTrack->charge()           // +1 or -1
muTrack->nHits()            // TPC hits
muTrack->nHitsFit()         // TPC hits used in fit
[...]
```

vectors and helices

```
muTrack->dcaGlobal()        // StThreeVectorF
muTrack->momentum()         // StThreeVectorF
muTrack->helix()            // StPhysicalHelixD
[...]
```

calculated pid quantities

```
muTrack->pidProbPion()      // 0. <= pidProb <= 1.0
muTrack->nSigmaKaon()       // nSigma (bethe-bloch)
[...]
```

MuEvent

```
StMuEvent *muEvent = mMuDstMaker->muDst()->event();
```

event-wise quantities

```
muEvent->refMult()
muEvent->primaryVertexPosition().z()
muEvent->magneticField()
muEvent->ctbMultiplicity()
```

detector- and trigger-wise collections

```
muEvent->triggerIdCollection()
muEvent->fpdCollection()
```

Instantiating StMuDstMaker

- StMuDstMaker can take file lists in many formats
  - Contents of a directory
  - Lists of files
  - Filename filtering possible

```
StMuDstMaker *muDstMaker = new StMuDstMaker( int mode,
                                              int nameMode,
                                              const char *dirName="./",
                                              const char *fileName="",
                                              const char *filter=".",
                                              int maxfiles=10,
                                              const char *name="MuDst" )
```

- Using the STAR scheduler is the best way to create the filelists:
  - Instantiate after creating an StChain:

```
StMuDstMaker *muDstMaker = new StMuDstMaker(0,0,"",inFile,"",100)
```

0 = read (1=write)

0 = read from inFile (1=ioMaker)

Max # .root files to chain

Name of .root file or filelist (pass to macro)

Using StMuDstMaker in a Macro

```
void MacroExample( Int_t nFiles, Char_t InputFileList[256] )
{
  // Load Libraries
  gROOT -> Macro("loadMuDst.C")
  // Create Histograms
  TCanvas*   nCanvas[2] ;
  TH1F*    histogram[2] ;
   …
  // Run the muDstMaker
  StMuDstMaker* maker = new StMuDstMaker(0,0,"",InputFileList,"MuDst",nFiles) ;
  while ( !maker->Make() )
    {
      // Do 'event' analysis based on event pointers
      StMuEvent* muEvent     = maker -> muDst()-> event() ;
      histogram[0] -> Fill( muEvent -> primaryVertexPosition().z() ) ;
      // Get 'track' data, make cuts on tracks, do physics analysis
      TObjArray* tracks = maker -> muDst()-> primaryTracks() ;
      TObjArrayIter GetTracks(tracks) ;
      StMuTrack* track ;
      while ( ( track = (StMuTrack*)GetTracks.Next() ) ) // Main loop for Iterating
        {
          histogram[1] -> Fill( track->pt() ) ;
        }
    }
  // Finish using the muDSTmaker
  …
  histogram[0] -> Draw() ;
  histogram[1] -> Draw() ;
}
```

This could be in the Init() function of a maker

Analyze 'event' data

Analyze 'track' data

This could be in the Finish() function of a maker

Full example at rcf: /star/institutions/lbl/jhthomas/muDstTutorial/MacroExample
Execute with root4star > MacroExample.C(1, "filename.MuDst.root")

Using StMuDstMaker in a Chain

```
void SimpleAnalysis(Int_t nEvents, Int_t nFiles, TString InputFileList,
                                   TString OutputDir, TString JobIdName )
{
  // Load libraries
  gROOT   -> Macro("loadMuDst.C")
  gSystem -> Load("SimpleAnalysis.so") ;

  // List of member links in the chain
  StChain*              chain    =  new StChain ;
  StMuDstMaker* muDstMaker       =  new StMuDstMaker(0,0,"",InputFileList,"MuDst",nFiles) ;
  SimpleAnalysisMaker* AnalysisCode  =  new SimpleAnalysisMaker(muDstMaker) ;

  …

  // Loop over the links in the chain
  chain -> Init() ;
  chain -> EventLoop(1,nEvents) ;
  chain -> Finish() ;

  // Cleanup
  delete chain ;
}
```

Precompile your analysis Maker using 'cons'

Your analysis Maker in the chain

```
Full example at rcf:  /star/institutions/lbl/jhthomas/muDstTutorial/SimpleExample
Execute with:  star-submit SimpleAnalysis.xml,  output into *.histograms.root file
Browse with root4star > new TBrowser("filename.histograms.root")
```

Example: SimpleAnalysisMaker

- The strategy for SimpleAnalysisMaker

	Constructor:	Pass pointer to StMuDstMaker object
	Init():	Instantiate & define histograms
every event	**Make()**:	Get pointers to StMuEvent & StMuTrack for current event, apply event cuts, loop over tracks, apply track cuts, & fill histograms
every event	**Clear()**:	Do nothing (some analyses may clear collections, etc.)
	Finish():	Write histograms to file, print information to stdout

SimpleAnalysisMaker.h

```
#ifndef SimpleAnalysisMaker_def
#define SimpleAnalysisMaker_def

#include "StMaker.h"
#include "TString.h"

class StMuDstMaker ;
class TFile        ;
class TH1F         ;

#define MaxNumberOfTH1F     10

class SimpleAnalysisMaker : public StMaker
{
 private:
  StMuDstMaker* mMuDstMaker ;
  TH1F*         histogram[MaxNumberOfTH1F] ;
  TFile*        histogram_output ;
  UInt_t        mEventsProcessed ;
  TString       mHistogramOutputFileName ;

 public:
  SimpleAnalysisMaker(StMuDstMaker* maker) ;
  virtual          ~SimpleAnalysisMaker( ) ;

  Int_t Init     ( ) ;
  Int_t Make     ( ) ;
  Int_t Finish   ( ) ;

  void SetOutputFileName(TString name) {mHistogramOutputFileName = name;}
  ClassDef(SimpleAnalysisMaker,1)
};

#endif
```

Unique to this analysis

Generic to all analyses

SimpleAnalysisMaker.cxx

```
ClassImp(SimpleAnalysisMaker)                       // Macro for CINT compatibility

SimpleAnalysisMaker::SimpleAnalysisMaker( StMuDstMaker* maker ) : StMaker()
 {
   for ( Int_t i = 0 ; i < NumberOfTH1F ; i++ )  {  histogram[i] = NULL ;  }
   mMuDstMaker       = maker ;                      // Pass MuDst pointer
   histogram_output = NULL   ;                      // Zero the Pointer to output file
   mEventsProcessed = 0      ;                      // Zero the Number of Events
   mHistogramOutputFileName = "" ;                  // Histogram Output File Name
 }


Int_t SimpleAnalysisMaker::Init( )
 {
   histogram_output = new TFile( mHistogramOutputFileName, "recreate" ) ;
   const Int_t nbins  =  100   ;
   histogram[0]  = new TH1F( "Vertex", "Vertex Z Position", nbins, -25.0, 25.0 ) ;
   histogram[1]  = new TH1F( "Pt", "Transverse Momentum", nbins, 0, 10 ) ;
   return kStOK ;
 }


Int_t SimpleAnalysisMaker::Finish( )
 {
   histogram_output -> Write() ;
   cout << "Total Events Processed in DstMaker " << mEventsProcessed << endl ;
   return kStOk ;
 }
```

Create histograms

Write histograms, close files, etc.

SimpleAnalysisMaker.cxx (continued)

```
Int_t SimpleAnalysisMaker::Make( )

{
  // Get 'event' data

  StMuEvent* muEvent       =  mMuDstMaker -> muDst()-> event()   ;
  histogram[0] -> Fill( muEvent->primaryVertexPosition().z() ) ;

  // Get 'track' data, make cuts on tracks, do physics analysis

  TObjArray* tracks = mMuDstMaker -> muDst()-> primaryTracks() ;
  TObjArrayIter GetTracks(tracks) ;                              // Create an iterator
  StMuTrack* track ;                                             // Pointer to a track

  while ( ( track = (StMuTrack*)GetTracks.Next() ) )             // Main loop for Iterating
    {
      histogram[1] -> Fill( track->pt() ) ;
    }

  mEventsProcessed++ ;
  return kStOK ;

}
```

Analyze 'event' data

Analyze 'track' data

BiggerAnalysisMaker.cxx

```
Int_t BiggerAnalysisMaker::Make( )

{
  // Get 'event' data
  StMuEvent* muEvent      =  mMuDstMaker->muDst()->event() ;
  histogram[0] -> Fill( muEvent->primaryVertexPosition().z() ) ;

  // Get 'track' data, make cuts on tracks, do physics analysis.
  TObjArray* tracks = mMuDstMaker->muDst()->primaryTracks() ;
  TObjArrayIter GetTracks(tracks) ;
  StMuTrack* track ;

  while ( ( track = (StMuTrack*)GetTracks.Next() ) )
    { histogram[1] -> Fill( track->pt() ) ; }

  // Option - you can store the tracks in a TList and do physics analysis
  // on the TList. This is a good way to do pairwise analyses.

  TList HighPtTracks ;
  GetTracks.Reset()  ;
  while ( ( track = (StMuTrack*)GetTracks.Next() ) )
    { if ( track->pt() >= 2.0 ) HighPtTracks.Add(track) ; }

  TListIter GetHighPtTracks(&HighPtTracks) ;
  StMuTrack* listedtrack ;
  while ( ( listedtrack = (StMuTrack*)GetHighPtTracks.Next() ))
    { histogram[2] -> Fill ( listedtrack->pt() ) ; }

  mEventsProcessed++ ;
  return kStOK ;
}
```

First pass at physics analysis

Create list and save selected tracks

Physics analysis on saved tracks

```
Full example at rcf:  /star/institutions/lbl/jhthomas/muDstTutorial/BiggerExample
Execute with: star-submit BiggerAnalysis.xml,  output into *.histograms.root file
        Browse with root4star > new Tbrowser("filename.histograms.root")
```

The Scheduler

- Q: Where do I get a list of files to run through my macro?
  - A: Don't

- STAR data files are so large that it is very inefficient to maintain a list of files and to run that list through a single CPU
  - The network is the bottleneck
  - For example: the run 4 DSTs use 40 TB of disk space and the disks will fail if you try to move a significant fraction of this around the farm

- It is better to move your job to the data
  - The STAR μDSTs are stored on local disks at RCF
  - You can send your job to the CPU where the data is stored
    - This is a very efficient use of the network because only your histograms have to come back over the network

- The STAR Scheduler does this for you automatically

The Scheduler builds the list of files

- The Scheduler builds the lists of files based on a query to the file catalogue to determine what you want. You can do this from the command line:

```
> get_file_list.pl -keys path,filename -cond filetype=daq_reco_mudst,
  trgsetupname=ProductionCentral,collision=AuAu200,sanity=1,storage=local
```

- Or you can do it with a script. A sample .xml file to coordinate the file catalogue query and the submission of your job might look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<job maxFilesPerProcess="10" fileListSyntax="xrootd" >

<command>
        starpro
        root4star -q -b SimpleAnalysis.C\(0,$INPUTFILECOUNT,\"$FILELIST\",\"$SCRATCH\",\"$JOBID\"\)
</command>

<SandBox installer="ZIP">
        <Package>
              <File>file:./SimpleAnalysis.C</File>
              <File>file:./StRoot</File>
        </Package>
</SandBox>

<input
  URL="catalog:star.bnl.gov?filetype=daq_reco_mudst,trgsetupname=productionCentral,sanity=1,storage=local”
  nFiles="2" />

<stdout URL="file:./$JOBID.out"/>
<stderr URL="file:./$JOBID.err"/>
<output fromScratch="*.root‟ toURL="file:/star/institutions/lbl/jhthomas/muDstTutorial/SimpleExample/" />

</job>
```

See the documentation on the STAR computing homepage => Grid => Scheduler

Putting it all together …

~home/SimpleAnalysisExample

SimpleAnalysis.xml, SimpleAnalysis.C, StRoot

~home/SimpleAnalysisExample/StRoot

SimpleAnalysis

~home/SimpleAnalysisExample/StRoot/SimpleAnalysis

SimpleAnalysisMaker.cxx, SimpleAnalysisMaker.h

- **In order to do STAR data analysis, you have 4 files to build**
  - **SimpleAnalysis.xml** ← **Query the file catalogue**
  - **SimpleAnalysis.C** ← **Put makers in a 'chain'**
  - **SimpleAnalysisMaker.cxx**
  - **SimpleAnalysisMaker.h**

  Do physics analysis

- **Submit your analysis request to the farm using this command**

  > star-submit SimpleAnalysis.xml

References

- This tutorial, its source code and a sample data file are saved in
  `rcf:  /star/institutions/lbl/jhthomas/muDstTutorial`
- STAR offline software is documented in CVS & D'Oxygen
  http://www.star.bnl.gov/webdata/dox/html/classes.html
  This link can be found from STAR homepage => computing => offline doc
- MuDsts by Dan Magestro (circa 2003)
  http://www.star.bnl.gov/public/comp/train/tut/MuDstTutorial/intro/MuDstTutorial.html
- Makers by Akio Ogawa (circa 2003)
  http://www.star.bnl.gov/public/comp/train/tut/StEventMaker/
- The Scheduler by Levente Hajdu
  http://www.star.bnl.gov/public/comp/Grid/scheduler/
- The StarClassLibrary by Thomas Ullrich
  http://www.star.bnl.gov/public/html/tmp/dev_StarClassLibrary.pdf
- Additional examples and food for thought are contained in example macros
  `rcf: $STAR/StRoot/macros/mudst   (after setting starver or starnew, etc.)`

Additional Topics

Cutting on the Data

Recommended cuts on the data for a 'typical' analysis

Cuts on event information:

muEvent->refMult()	// Centrality determination
muEvent->primaryVertexPosition()	// Not equal to (0,0,0) for older daq files
mMuDstMaker->muDst()->numberOfPrimaryVertices()	// == 0,1 for new files

Cuts on track quality:

muTrack->dcaGlobal().mag()	<	2 cm
muTrack->eta()	<	1.0
muTrack->flag()	≥	0
muTrack->nHitsFit()	>	15
nHitsFit() / nHitsMax()	>	0.55

For additional details about the data, trigger conditions, etc:
http://www.star.bnl.gov/STAR/comp/prod/
especially the trigger information on that page. Also see:
http://www.star.bnl.gov/protected/common/common2004/trigger2004/200gev/200gevFaq.html
for 2004 run information. Move higher up in this directory tree to see more information about other years trigger and centrality information. But most important … talk to your colleagues to get their advice!

The StarClassLibrary

- **Note that the μDST files use structures that are unique to STAR**
  - **StThreeVectorF**

    is a three vector that can be used to take dot products, cross products and several other very useful functions
  - **StPhysicalHelixD**

    is a state vector containing the parameters of a helix
  - **Etcetera, etcetera**

- **These are classes from the StarClass Library**
  - **SCL is a very comprehensive library of tools and beautifully written**

    http://www.star.bnl.gov/public/html/tmp/dev_StarClassLibrary.pdf

- **ROOT has equivalent classes that do the same thing**
  - **TVector3 is essentially identical to StThreeVectorF, etc.**
  - **ROOT was a primitive beast at the time the StarClass Library was written and it did not have ThreeVectors.   So we wrote our own.**

There is no way to change the fact that we use the StarClassLibrary and so you must have it, and use it, in order to analyze STAR data.