What Bugs me about Milagro Software?

How many people have written sky mapping code?

How many people have written bad event filtering code?

How many people know how to write their own Reconstructed data files?

How many people know how to strip raw events?

How many people can easily search the sky for DC point sources?

How many people can search the sky for a GRB at a known position and time?

A “Framework” for the Milagro Software

The Problem:

- The user writes/owns the main.
- Minimal examples are >100 lines
- Users have lots of control (Good).
- The system does not provide services for the user (Bad)
- No common framework makes it hard for users to share code (also Bad)
- No common framework means code “fixes” might not extend to users code (really really Bad).

What should be standard resources are difficult to use or not included:

- Data writing routines RAW/REC
- Filtering code
- Map making code
- Histogram making code

A “Framework” for the Milagro Software

What the code does:

- Provides a standard main() versatile enough for all users (hopefully).

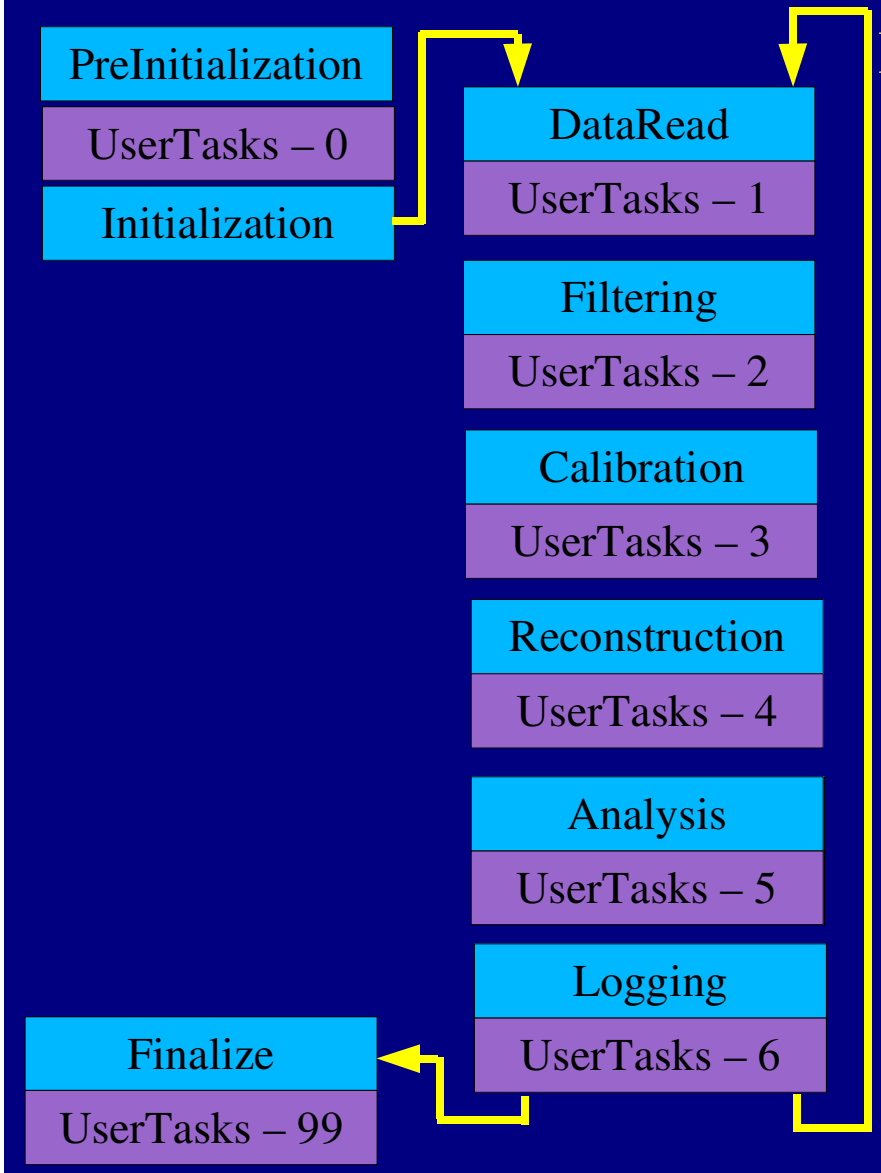
- Provide a uniform interface for access to Raw,Reconstructed and Monte Carlo Data.

- Provides a standard interface for filter.

- Provides a simple utilities for Logging data.

- A whole bunch of other stuff, As much as people want to add.

How the Framework works



Event Loop

The function UserTasks() is called after every stage. Based on the results of the system at the end of each stage, the user can choose to steer the event how they wish.

UserTasks() receives pointers to 2 structures:

UserTasks(EVENT *, STEER *) { }

EVENT - All the familiar Milagro data structures

STEER - Steering switches for framework

EVENT Structure

```
typedef struct
{
  EVENT_DATA        eventData;
  ANGLE_RECON       AngleFit;
  CORE_RECON        CoreFit;
  MUON_RECON        Muon;
  GH_RECON          ghInfo;
  GRID_COORD        coords;
  PMT_INFO          pmtInfo;
  ANGLE_CONTROL     angleCon;
  CORE_CONTROL      coreCon;
  GH_CONTROL        ghCon;
  DATA_CONTROL      dataCon;
  CAL_STATUS        calStatus;
  EDGE_INFO         edgeInfo;
  CAL_CONTROL       calCon;
  POOL_RECON        poolRecon;
} EVENT;
```

STEER Structure

```
/* steering routines for Milagro main() program */
typedef struct
{
  int CurrentStage;    /* Current Stage of Reconstruction: init=0, 1-3 processing, 99=end  */
  int Continue;        /* User Control Structure, 1=continue to next level, 0=next event   */
  int RepeatEvent;     /* Repeat Calibration/Reconstruction cycle for current event        */
  int TerminateLoop;   /* Terminate Program, call finalize routine.                        */
  int LogRecEvent;     /* set to 1 if Reconstructed event should be written to a file      */
  int LogRawEvent;     /* set to 1 if Raw event should be written to an output file        */
  int MonitorActive;   /* Turn on Monitor                                                  */
  int DisplayEvent;    /* Call event Display for this event                                */
  int SpecialArgs;     /* Arguments passed from command line indicate filename and Nevents */
  int Verbose;         /* If 1 print statistics and stuff                                  */
  int ComputeSensi;    /* Compute point source sensitivity                                 */
  int MilagroEventStatus;    /* Status variable returned by MilagroEvent4                  */
  int StageCount;      /* Number of stages in Event Loop                                   */
  int RecEventsLogged; /* =0 if no Rec Events were logged, =1 if Rec Events were Logged    */
  struct {
    float SpectralIndex,       /* Spectral index of desired spectrum          */
          I0,                  /* Source Amplitude gammas/m**2/Tev            */
          SourceDec,           /* Source declination                          */
          TypicalRadius,       /* Typical Radius of MC showers that pass cuts */
          EMinMC,              /* Minimum Energy of MC data                   */
          EMaxMC,              /* Maximum Energy of MC data                   */
          SpectralIndexMC,     /* Spectral Index of MC data                   */
          MCThrowRadius;       /* MC throw radius. flat R distriubtion assumed */
    char  SensiFilename[100]; /* Filename of log file                         */
  } SensiParameters;  /* Steering parameters for point source sensi program              */
  int argc;           /* number of parameters passwd from command line                   */
  char *argv[4];      /* command line parameters; GNUstep wants the array size specified  */
} STEER;
```

Added Software

Point source Sensitivity program:
Calculates events/day from the Crab.

Monitor:
The monitor program used online is now available at the touch of a switch

Simple Event Logging:
Log Reconstructed or Raw data with a simple switch.

Monitor Example

```
/*

   This is an example of how to call the Milagro data Monitor
   program using the user filter interface.
   09/11/93 -AJS

*/

#include "Reconstruction.h"
#include "Offline.h"

int UserTasks( EVENT *Event, STEER *Steer ) {
  static int nEvents=0;

  Steer->Continue = 1;       /* always continue to next stage */

  if (Steer->CurrentStage==0) {   /* initialization */
    Steer->MonitorActive=1;
  }

  if (Steer->CurrentStage==1) {
    nEvents++;
    if (nEvents % 1000 == 0) printf ("Monitor: %8d Events Read\n",nEvents);
  }

  if (Steer->CurrentStage==99) {
    printf ("Monitor: %8d Events Read - Finished\n",nEvents);
  }


}
```

Reconstruction Example

```
/*
   This is an example of how to reconstruct data with the Milagro code`
  and write out the REC files with the results of the reconstruction.
  09/11/93 -AJS
*/
#include "Reconstruction.h"
#include "Offline.h"

int UserTasks( EVENT *Event, STEER *Steer ) {
 static int nEvents=0;

 Steer->Continue = 1;       /* always continue to next stage */

 switch (Steer->CurrentStage) {

 case 0:  /* Initialization */
  Steer->MonitorActive=0;
  Steer->LogRecEvent = 1;
  break;

 case 1:  /* Event read */
  nEvents++;
  if (nEvents % 1000 == 0) printf ("Reconstruction: %8d Events Read\n",nEvents);
  break;

 case 99: /* Finalize */
  printf ("Recon: %8d Events Read - Finished\n",nEvents);
  break;

 default: /* Everything else */
  break;

 }  /* end switch */

}
```

Stuff the Framework Still Needs

- Message Logging Interface (people can reroute standard messages)
- GetArgs() command line interface:

```
$ ./milagro -?                  # show usage
$ ./milagro -n 1000             # process 1000 events
$ ./milagro -j 2345             # run on data from julian date 2345
$ ./milagro -r 5000 -s 1-10     # run on data from  run 5000, sr 1-10
$ ./milagro -u 1 2 3.4 “abc.map”  # user parameters
```

- Set up a standard place for the current libs at LANL and UCSC with nightly builds. Build date in code?
- More and better examples: Maps making
- Configuration banner
- Set of standard statistics: events passing each level.
- Integrated Event Display
- Integrated DB Interface
- Sub Run Header decoder and printer outer.