



Writing a DMT Monitor

John G. Zweizig

LIGO / Caltech

LHO, August 21, 2004



Monitor Functional Overview

- A monitor must do the following:
 - » Get input frame data
 - » Perform algorithm
 - Signal processing
 - Measure some physical quantity
 - Make consistency checks
 - » Publish statistical summary
 - Html status page
 - Trend data
 - Publish result objects
 - » Generate triggers



Monitor Structure

- Environment class provides
 - » Set up data access
 - » Iterate over data strides
 - » Handle error conditions
 - » Clean termination.
- Monitor author provides virtual methods
 - » Initialization (monitor class constructor)
 - » Stride-by-stride processing (ProcessData())
 - » Signal handling (Attention())
 - » Monitor termination (destructor)



Monitor Template Classes

- Template classes available in <http://www.ldas-sw.caltech.edu/cgi-bin/cvsweb.cgi/gds/DMT/Templates?cvsroot=GDS>
- or `$DMTHOME/$DMTVERSION/etc/templates`
- Single Channel class is in `DatTemplate.{cc,hh}`
- Multi-Channel class is in `DatTemplateMC.{cc,hh}`
- Monitor function
 - » Filter Time series of one/many input channel(s)
 - » Calculate rms of filtered channel(s)
 - » Keep history, serve time series up to 12 hours
 - » Write trend of rms values



Define the Monitor

```
#include "DatEnv.hh"
#include "MonServer.hh"
#include "FixedLenTS.hh"
#include "Trend.hh"
#include <string>

class DatTemplate : public DatEnv, MonServer {
public:
    DatTemplate(int argc, const char *argv[]); // ctor
    ~DatTemplate(void); // Destructor
    void ProcessData(void); // Process data stride
    void Attention(void); // Handle signals.
private:
    int maxFrame; // frames to process
    Interval mStep; // Time stride.
    Trend mTrend; // Trend writer
    std::string mChannel; // Channel name
    std::string mFilter; // Filter description
    Interval mSettle; // Settling time.
    std::string mSigmaName; // trend name
    Pipe* mPipe; // Pipe pointer
    FixedLenTS mHistory; // X sigma history.
};
```

- Define monitor class
 - » Based on DatEnv class
 - » monitor data server
 - » Trend writer
- Virtual methods
 - » Constructor (initialization)
 - » Destructor (termination)
 - » ProcessData
- Data Values
 - » Channel data: name
 - » Processing: stride, filter def, settling time, trend name
 - » Status: filter, history, trend



Where's the main Function?

```
#include "DatTemplate.hh"
```

```
#define PIDCVSHDR "$Header: $"
```

```
#define PIDTITLE "Data monitor template"
```

```
#include "Proclident.hh"
```

```
using namespace std;
```

```
EXECDAT(DatTemplate)
```

- Proclident.hh

- » Generates process identifier (static object)
- » PIDCVSHDR: version id filled in automatically by cvs
- » PIDTITLE: monitor title

- EXECDAT main program C preprocessor macro

- » Construct monitor class object: pass command-line arguments
- » Invoke monitor::MainLoop()
- » Destroy object on completion



Initialization: Parse Command

```
DatTemplate::DatTemplate(int argc, const char *argv[])
: DatEnv(argc, argv), maxFrame(999999), mStep(2.0),
  mPipe(0), mHistory(43200)
{
  bool dvIndex = false, syntax = false;
  for (int i=1 ; i<argc ; i++) {
    string argi = argv[i];
    if (argi == "-channel") mChannel = argv[++i];
    else if (argi == "-dvindex") dvIndex = true;
    else if (argi == "-filter") mFilter = argv[++i];
    else if (argi == "-frame")
      maxFrame=strtol(argv[++i], 0, 0);
    else if (argi == "-stride")
      mStep = strtod(argv[++i], 0);
    else if (argi == "-settle")
      mSettle = strtod(argv[++i], 0);
    else if (isDatEnvArg(argv[i])) i++;
    else syntax = true;
  }
}
```

- DatEnv(int, const char**) constructor must be used
- Loop over command line arguments
- isDatEnvArg() is true for arguments handled by DatEnv constructor



Initialization: Setup

```
//----- Force a valid channel
if (mChannel.empty()) {
    mChannel = "H0:PEM-LVEA_SEISX";
}
//----- set the data accessor
getDacc().setStride(mStep);
getDacc().addChannel(mChannel.c_str());

//----- Specify DMT Viewer channels
char username[32];
userid(username);
setServerName(username);
mSigmaName = mChannel + "_sigma";
serveData(mSigmaName.c_str(), &mHistory, 0);

//----- Set up the trender
mTrend.setName("DatTemplate");
mTrend.setFrameCount(1);
mTrend.setType(Trend::kMinute);
mTrend.setAutoUpdate(false);
mTrend.addChannel(mSigmaName.c_str());
```

- Set data accessor stride and channel name.
- Set monitor data server name
- Set viewer/trend channel name
- Set up dmtviewer object
- Set up trend writer... add a channel



Process 1 Stride

```
void
DatTemplate::ProcessData(void) {
    //----- Get pointers to the current data.
    const TSeries* ts = getDacc().refData(mChannel.c_str());
    if (!ts || ts->isEmpty()) {
        cout << "Channel: " << mChannel << " was not found." << endl;
        return;
    }

    //----- Construct the filter if it isn't already there
    if (!mPipe && !mFilter.empty()) {
        double samplerate(1./double(ts->getTStep()));
        FilterDesign fd(mFilter.c_str(), samplerate);
        mPipe = fd.release();
    }

    //----- Filter the data
    TSeries tf;
    if (mPipe) tf = mPipe->apply(*ts);
    else     tf = *ts;

    //----- Calculate channel sigma.
    float Sig = sqrt( tf*tf/tf.getNSample() - pow(tf.getAverage(), 2) );

    //----- Trend the data points
    Time t0 = tf.getStartTime();
    mTrend.trendData(mSigmaName.c_str(), t0, Sig);
    mTrend.Update();

    //----- set up data for display.
    mHistory.fixedAppend(t0, tf.getInterval(), &Sig);
}
```

- Get input TSeries pointer
- Design a filter if needed
 - » Sample rate from ts
 - » Filter defined by arg
- Filter the Data
- Calculate sigma (rms)
- Trend rms
- Save rms in history



Termination

```
//===== Handle Signals
void
DatTemplate::Attention(void) {
    MonServer::Attention();
}

//===== Object destructor.
DatTemplate::~DatTemplate()
{
    cout << "DatTemplate is finished"
         << endl;
}
```

- Signal handler
 - » Pass signals to monitor server
- Termination handler (destructor)
 - » Delete allocated variables
 - » Print job summary
 - » Close I/O files



Multi-Channel Monitors

- Multi-Channel monitors require the following extensions:
 - » Define channel descriptor class and channel list
 - » Read channel configuration from file
 - » Loop over list of channels for each stride.



Defining a Channel List

```
class chan_data {
public:
    chan_data(const std::string& chan);
    chan_data(const chan_data& c);
    ~chan_data(void);
    float computeSigma(const TSeries& ts);
    const char* getChannel(void) const;
    const char* getSigmaName(void) const;
    const TSeries& refHistory(void) const;
    void setFilter(const std::string& f);
    void setSettle(Interval dt);
private:
    std::string mChannel; // Channel name
    std::string mFilter;   // Filter description
    Interval mSettle;     // Settling time.
    std::string mSigmaName; // trend channel
    Pipe* mPipe;         // Pipe pointer
    FixedLenTS mHistory; // X sigma history.
};

typedef std::vector<chan_data> chan_list;
typedef chan_list::iterator chan_iter;
typedef chan_list::const_iterator const_chan_iter;
```

- Define a channel class
 - » Constructor(s)
 - » Computation method
 - » Accessors (getName(), etc.)
 - » Set parameters.
- Move channel definition and status parameters to channel class
- Stl container to hold list of channels
 - » `std::vector<>` is easy and efficient
 - » Usually need iterators.



Read Channel Configuration

```
void
DatTemplateMC::configure(const char* file) {
    ParseLine pl(file);
    bool syntax = false;
    while (pl.getLine() >= 0) {
        int nArg = pl.getCount();
        string Chan = pl[0];
        chan_data ch(Chan);
        for (int i=1; i<nArg; ++i) {
            string argi = pl[i];
            if (argi == "-filter") {
                ch.setFilter(pl[++i]);
            } else if (argi == "-settle") {
                ch.setSettle(pl.getDouble(++i));
            } else {
                syntax = true;
            }
        }
        mList.push_back(ch);
    }
}
```

- Read in configuration
- Add each new channel to end of list



Processing One Channel at a Time

```
void
DatTemplateMC::ProcessData(void) {
    int N = mList.size();
    for (int i=0; i<N; ++i) {
        const TSeries* ts =
            getDacc().refData(mList[i].getChannel());
        if (!ts || ts->isEmpty()) continue;
        float Sig = mList[i].computeSigma(*ts);
        Time t0 = ts->getStartTime();
        mTrend.trendData(mList[i].getSigmaName(), t0, Sig);
    }
    mTrend.Update();
}
```

```
float
chan_data::computeSigma(const TSeries& ts) {
    if (!mPipe && !mFilter.empty()) {
        double samplerate(1./double(ts.getTStep()));
        FilterDesign fd(mFilter.c_str(), samplerate);
        mPipe = fd.release();
    }
    TSeries tf;
    if (mPipe) tf = mPipe->apply(ts);
    else      tf = ts;
    float Sig = sqrt( tf*tf/tf.getNSample()
        pow(tf.getAverage(), 2) );
    mHistory.fixedAppend(tf.getStartTime(), tf.getInterval(),
        &Sig);
    return Sig;
}
```

- Loop over channels in ProcessData
- Call processing method of channel class.