# Calibration API in JANA

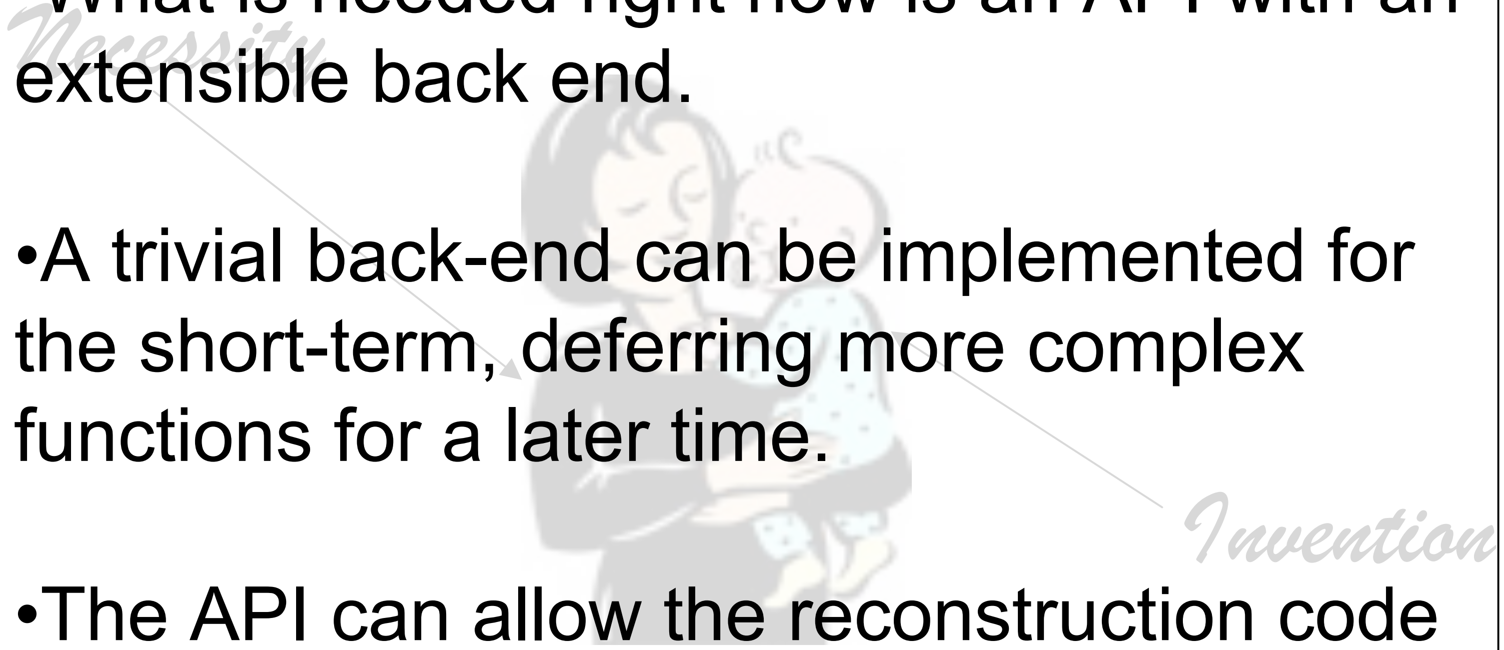
David Lawrence, Jlab

July 17, 2007

## Previous Work

- Detailed discussions were taking place by May 2005 on the desired feature set for the calibration database
- A committee was formed consisting of: Nikolay Kolev, Mark Ito, Greg Riccardi, and David Lawrence whose task was to come up with an initial design
- GlueX-doc 672: “Hall D Calibration Database Table Design and interface” by Nikolay Kolev
- Nothing was ever implemented into JANA/DANA

# What do we need right now?



- What is needed right now is an API with an extensible back end.

- A trivial back-end can be implemented for the short-term, deferring more complex functions for a later time.

- The API can allow the reconstruction code to be developed without fear of needing to change it all “later”.

# The API

Calibration constants will be accessed through a *GetCalib(…)* method of the JEventLoop class

```
// Get 1-D array of values indexed by name
bool GetCalib(string namepath, map<string, T> &vals)

// Get  1-D array of values indexed by row
bool GetCalib(string namepath, vector<T> &vals)

// Get 2-D table of values indexed by row and name
bool GetCalib(string namepath, vector< map<string, T> > &vals)

// Get 2-D table of values indexed by row and column
bool GetCalib(string namepath, vector< vector<T> > &vals)
```

## The JCalibration Class

- An instance of JCalibration represents a single calibration across the entire detector
- JCalibration objects contain a min and max run number for which they are valid
- JApplication creates the JCalibration objects as needed and keeps them in a list to service other requests
- Users typically won’t deal with JCalibration directly, but will access it through JEventLoop

## Specifying the “namepath”

The *namepath* is a single string representing a hierarchical path to a set of named constants.

Example:

**CDC/timewalk/stereo_par**

## Accessing the Constants

*... in factory class definition ...*

```
vector<int> peds;
```

*... in brun() method ...*

```
loop->GetCalib("FCAL/pedestals", peds);
```

*... in evnt() method ...*

```
hit->ADC -= peds[hit->id];
```

# Accessing the Constants

*... in factory class definition ...*

```
double slope, offset, exponent;
```

*... in brun() method ...*

```
map<string, double> twpars;
loop->GetCalib("FDC/driftvelocity/timewalk_parameters", twpars);

slope    = twpars["slope"];
offset   = twpars["offset"];
exponent = twpars["exponent"];
```

## File Formats

- ASCII files should have one “entry” per row
- Empty lines and lines beginning with “#” are ignored*
- “Keys” are optional

## Example 1

Simple 1-D array without keys. Six values will be read from this file and will be indexed by 0-5 if read into a vector or by the strings "0", "1", "2", ... if read into a map.

```
# This line is a comment and will be ignored
37
43
56
# This line will also be ignored.
22
63
38
```

## Example 2

Simple 1-D array with keys. Six values will be read from this file and will be indexed by 0-5 if read into a vector or by the strings "mean1", "sigma1", "offset1", ... if read into a map.

```
 # This line is a comment and will be ignored
mean1   37
sigma1  43
offset1 56
mean2   22
sigma2  63
offset2 38
```

# Example 3

Table with keys. In order to specify keys in a table, one uses the special "#%" syntax. When read into a vector<map<string,T>>, the white-space-separated values on the #% line are used to index the map part.

```
 #% x z      bx bz     nx        nz
0 210.82 0 -2.1816 -0.328271  0.000871416
0 213.36 0 -2.1829 -0.329189 -0.000873593
0 215.9  0 -2.1842 -0.329705 -0.00100867
0 218.44 0 -2.1855 -0.329103  0.00163991
0 220.98 0 -2.1868 -0.328649  0.00188895
0 223.52 0 -2.1882 -0.328796  0.00165464
...
```

## Specifying the Location of the Calibration Database

- The location is determined by the JANA_CALIB_URL environment variable

Example:

JANA_CALIB_URL is **file:///home/davidl/HallD/calib**

namepath is **FCAL/pedestals**

Put constants in the file:

**/home/davidl/HallD/calib/default/FCAL/pedestals**

## Summary

- Support for using calibration constants exists in the latest revision of JANA and can be used right now
- The API should support a much more advanced calibration database to be implemented in the future