Recent Developments on JCalibration

Feb. 11, 2009

David Lawrence JLab

Calibration Constants

Design criteria:

• **B-coders agnostic to storage mechanism**
Don’t care if they are retrieved from a database, file, web object, …

• **B-coders agnostic to calibration context**
Implicitly want “what everybody else is using” (e.g. same run number, same source, same “tag”, …)

B-coder is person writing reconstruction code

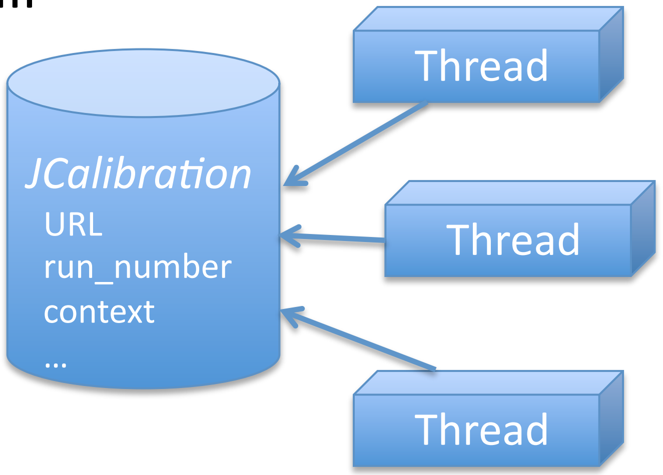
JCalibration

- A complete calibration is represented by a single *JCalibration* object that is shared by all threads

- One job may have multiple JCalibration objects (e.g. multiple runs in the job)

- Calibration source is specified by a URL (environment variable *JANA_CALIB_URL*)

- Factories don't specify calibration context (e.g. run number), it is already known by *JEventLoop*



JCalibration API

While we don’t actually have a calibrations/conditions database yet, we do have a well-defined API for accessing it.

Constants can be stored in either arrays (1D) or tables (2D) and can be indexed either by name (key-value) or by position.

<u>Templated methods of *JEventLoop*:</u>

arrays

```
// 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)
```

tables

```
// 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)
```

Example of Accessing Calibration Constants as key-value pairs

... 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"];
```

Template method converts values to doubles using stringstream class

For a few parameters like this, it makes sense to copy them into local data members of the factory class

Example of Accessing Calibration Constants as an array

... in factory class definition ...

```
vector<double> tof_tdc_offsets;
```

... in brun() method ...

```
loop->GetCalib(”TOF/tdc_offsets", tof_tdc_offsets);
if(tof_tdc_offsets.size()!=Ntof) throw JException(“Bad Ntof!”);
```

... in evnt() method ...

```
 double t = tof->tdc - tof_tdc_offsets[tof->id];
```

Backend Database

- The API defines the routines B-coders will use to obtain calibration constants independent of the details of how the actual database is implemented
- This does impose some requirements of the database design itself:
  - Store both 1-D arrays and 2-D tables
  - Index either by name or position
  - Uniquely identify constants by
    - Run number
    - Context string (may include timestamp)
    - URL
- *JCalibrationFile* implements a trivial calibration backend that maps directly to ASCII files on the local file system
  - Represents snapshot of constants and so ignores run number and context string
  - URL points to root directory (e.g. file:///group/halld/calib)
  - Constants currently kept in svn (https://halldsvn.jlab.org/repos/trunk/calib)

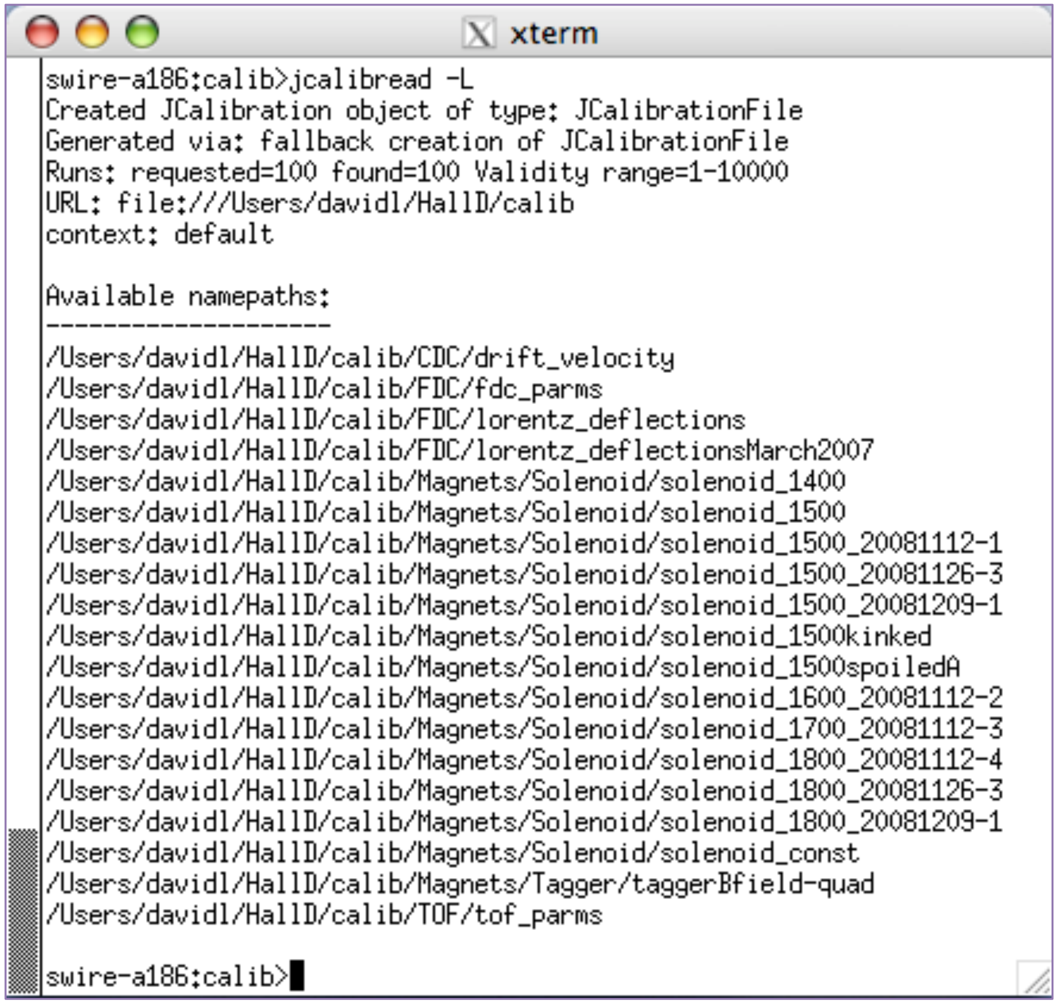
New Features

Implemented in JANA 0.4.9

Calibration object generators and *namepath* Discovery

- **Generator mechanism for *JCalibration***
  - *JCalibrationGenerator* class added
  - Allows multiple types of database backends to be supported in same binary
  - Allows new calibration database backends to be added dynamically to pre-compiled binaries
  - Useful for private development of backend alongside trunk without disturbing standard builds

- **Discovery mechanism**
  - *JCalibration* now has a new virtual method called *GetListOfNamepaths*() that can be used to probe a calibration backend for the available constants
  - This is utilized in the *jcalibread* utility using the "-L" switch

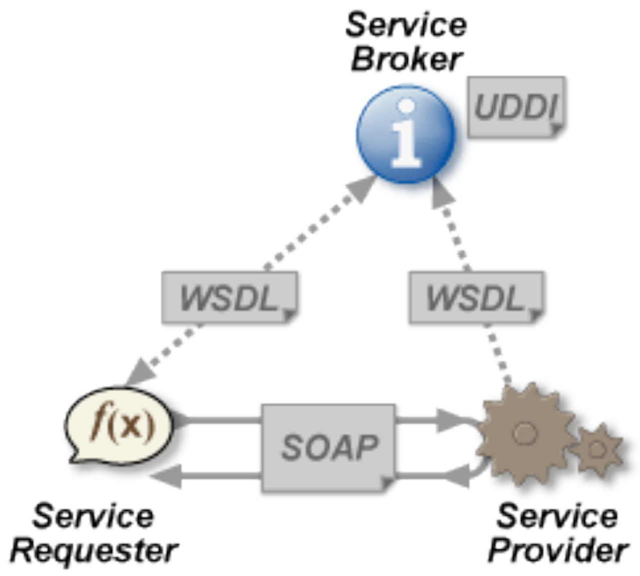


```
swire-a186:calib>jcalibread -L
Created JCalibration object of type: JCalibrationFile
Generated via: fallback creation of JCalibrationFile
Runs: requested=100 found=100 Validity range=1-10000
URL: file:///Users/davidl/HallD/calib
context: default

Available namepaths:
--------------------
/Users/davidl/HallD/calib/CDC/drift_velocity
/Users/davidl/HallD/calib/FDC/fdc_parms
/Users/davidl/HallD/calib/FDC/lorentz_deflections
/Users/davidl/HallD/calib/FDC/lorentz_deflectionsMarch2007
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1400
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500_20081112-1
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500_20081126-3
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500_20081209-1
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500kinked
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1500spoiledA
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1600_20081112-2
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1700_20081112-3
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1800_20081112-4
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1800_20081126-3
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_1800_20081209-1
/Users/davidl/HallD/calib/Magnets/Solenoid/solenoid_const
/Users/davidl/HallD/calib/Magnets/Tagger/taggerBfield-quad
/Users/davidl/HallD/calib/TOF/tof_parms

swire-a186:calib>
```

Calibration Web Service



• Calibration constants will need to be accessible from remote computers via the internet

• Direct access to a database is problematic due to cybersecurity concerns

• Web services work over HTTP and so will likely be the appropriate mechanism for remote access

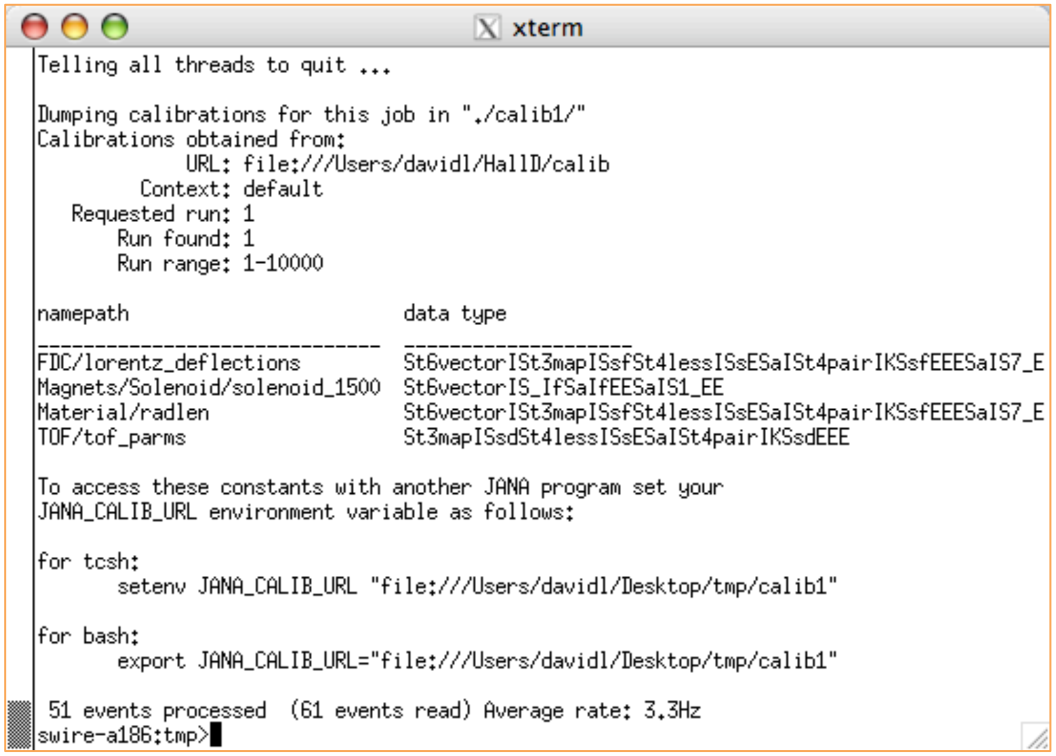
- The *JCalibrationWS* class has been written to provide calibration constants through a web service
  - Implemented as a plugin so --*jcalibws* must be added to command line to access (for now)
  - Allows read-only access to Hall-D calibration constants from anywhere in the world over HTTP (http://www.jlab.org/Hall-D/Software/test/calib)
  - Uses gSOAP, a C++ SOAP implementation
  - Currently works like a proxy for JCalibrationFile on server side, but could trivially be made to use another type of backend

Saving a (semi-)complete set of calibration constants to the local disk

All JANA programs now have a new command line option:

--dumpcalibrations

- Records which namepaths are requested during a job and writes the constants into ASCII files compatible with *JCalibrationFile*
- Avoids copying and running entire database or even copying a “complete” set of calibration constants (which could include obsolete ones or ones not applicable to the current run/code version)



```
Telling all threads to quit ...

Dumping calibrations for this job in "./calib1/"
Calibrations obtained from:
             URL: file:///Users/davidl/HallD/calib
         Context: default
   Requested run: 1
       Run found: 1
       Run range: 1-10000

namepath                        data type

------------------------------  --------------------
FDC/lorentz_deflections         St6vectorISt3mapISsfSt4lessISsESaISt4pairIKSsfEEESaIS7_E
Magnets/Solenoid/solenoid_1500  St6vectorIS_IfSaIfEESaIS1_EE
Material/radlen                 St6vectorISt3mapISsfSt4lessISsESaISt4pairIKSsfEEESaIS7_E
TOF/tof_parms                   St3mapISsdSt4lessISsESaISt4pairIKSsdEEE

To access these constants with another JANA program set your
JANA_CALIB_URL environment variable as follows:

for tcsh:
       setenv JANA_CALIB_URL "file:///Users/davidl/Desktop/tmp/calib1"

for bash:
       export JANA_CALIB_URL="file:///Users/davidl/Desktop/tmp/calib1"

 51 events processed  (61 events read) Average rate: 3.3Hz
swire-a186:tmp>
```

Recycled Containers

A new templated *Get()* method has been added to *JCalibration* that instructs it to keep ownership of the constants and just return a `const` pointer to the container.

Since STL vectors keep internal data sequential in memory, the values can be accessed via a standard array pointer while maintaining `const` correctness.

... in factory class definition ...

```
 const double *fcal_gains;
```

... in brun() method ...

```
const vector<double> *my_fcal_gains;
loop->GetCalib(”FCAL/Energy/gains", my_fcal_gains);
fcal_gains = &(my_fcal_gains->front());
```

... in evnt() method ...

```
double Ecorr = fcal_hit->E * fcal_gains[fcal_hit->id];

fcal_gains[3] =1.2; // This will generate compile time error!
```

Summary

- JANA's calibration database API can be used now to develop code using calibration constants kept in ASCII files. Code will not need to be changed once a "real" database is created for the backend.
- A proof-of-principle web service has been created for accessing calibration constants over the web. This will likely be deployed in the next couple of months for general use.
- The --dumpcalibrations switch has been added to all JANA programs allowing a snapshot of the constants used to be stored locally and re-used on subsequent jobs.
- Global storage (container recycling) has been added to the JCalibration base class reducing the potential memory footprint as well as potentially improving access speed.