JANA 0.6.0

David Lawrence JLab

Jan. 12, 2010

JStreamLog and *JStreamLogBuffer*

- Now have *jout* and *jerr* defined
  - Thread-safe (won't intermix strings like *cout/cerr*)
  - Optional prefixes can be set for each stream
    - Prefix string
    - Automatic timestamp (1 sec accuracy)
    - Thread id
- Any number of JStreamLog objects can be defined (e.g. for FDC)

```
>jana nothing
JANA ERROR>>Unable to open event source!
JANA >>Launching threads .
JANA >>
JANA >>No more event sources
JANA >>Thread 0xb0103000 completed gracefully
JANA >> 0 events processed  (0 events read) Average rate: 0.0Hz
```

JStreamLog output options

- `JANA:JOUT_TAG >`
- `JANA:JOUT_TIMESTAMP_FLAG 1`
- `JANA:JOUT_THREADSTAMP_FLAG 1`
- `JANA:JERR_TAG ERROR >>`
- `JANA:JERR_TIMESTAMP_FLAG 1`
- `JANA:JERR_THREADSTAMP_FLAG 1`

sample output …

```
Tue Jan 12 13:09:48 2010 # thr=0xa088e720 # >Created JCalibration object of type: JCalibrationFile
Tue Jan 12 13:09:48 2010 # thr=0xb0081000 # ERROR >>Unable to open event source!
Tue Jan 12 13:09:48 2010 # thr=0xa088e720 # >Generated via: fallback creation of JCalibrationFile
Tue Jan 12 13:09:48 2010 # thr=0xa088e720 # >Runs: requested=1 found=1 Validity range=1-10000
Tue Jan 12 13:09:48 2010 # thr=0xa088e720 # >URL: file:///Users/davidl/HallD/calib
Tue Jan 12 13:09:48 2010 # thr=0xa088e720 # >context: default
```

JCalibration now has *PutCalib()* method

- The JCalibration object provides an API for accessing calibration/conditions constants
  - JCalibrationFile (currently used. Based on simple ascii files)
  - JCalibrationMySQL (future)
- PutCalib method provides mechanism to write constants to JCalibration sub-classes that support writing

jcalibcopy utility

- The *jcalibcopy* utility can read from any calibration source and write to any
  - Read from file, write to Database
  - Read from Database, write to file,
  - Read from one Database, write to another,
  - …
- Users can:
  - Copy constants from database to local filesystem (avoid database accesses at every invocation)
  - Edit and test constants locally in ASCII files and copy them into database once they are ready

jcalibcopy utility

```
>jcalibcopy
Usage:
       jcalibcopy [options] -R src_run -Rmin dest_run_min -Rmax dest_run_max namepath src_url dest_url

Copy item from one JANA data source to another.

Required:

     -R src_run          Set run number for source
     -Rmin dest_run_min  Set min run number for destination
     -Rmax dest_run_max  Set max run number for destination

Options:
     -T                  Data set is a 2-D table (def. assume 1-D array)
     -a author           Set author (def. use USER env. var.)
     -h                  Print this message

 The url given for either the source or the destination may include a
context by appending "@context" to it.

 e.g.
   file:///home/billybob/calib@test1
```

example:

```
jcalibcopy   Magnets/Solenoid/solenoid_const   $JANA_CALIB_URL   file://$PWD/calib
```

GetMultiple() method added to JGeometry

- Geometry information is obtained from the XML using an *xpath* syntax

… snippet from CentralDC_HDDS.xml …

```
<composition name="CentralDC">
      <posXYZ volume="centralDC_option-1" X_Y_Z="0.0  0.0  71.71" />
</composition>
…
<tubs name="STRA" Rio_Z="0.000  0.800  150." material="CDchamberGas” sensitive="true" />
```

```
dgeom->Get("//posXYZ[@volume='CentralDC']/@X_Y_Z",cdc_origin);
dgeom->Get("//tubs[@name='STRA']/@Rio_Z",cdc_length);
```

- Previous implementation was limited to a single set of values corresponding to the first node found that matched the xpath

GetMultiple() method added to JGeometry

Start Counter definition has several polyplane elements, all with same xpath (excluding using the numbers)

```
<pgon name="STRT" segments="40" material="Air"  comment="start counter assembly”>
        <polyplane Rio_Z="6.911305 8.937551 -140.010000" />
        <polyplane Rio_Z="6.911305 8.937551 49.990000" />
        <polyplane Rio_Z="6.907519 8.937551 50.051907" />
        <polyplane Rio_Z="6.884555 8.914588 50.427356" />
        <polyplane Rio_Z="6.838703 8.899456 50.674755" />
        <polyplane Rio_Z="6.804705 8.865458 50.858193" />
        <polyplane Rio_Z="6.672943 8.788006 51.276087" />
        <polyplane Rio_Z="6.639577 8.774437 51.349302" />
        <polyplane Rio_Z="6.491236 8.671805 51.674808" />
        <polyplane Rio_Z="6.289760 8.568142 52.003585" />
        <polyplane Rio_Z="6.262290 8.547713 52.048412" />
        <polyplane Rio_Z="5.856563 8.283648 52.627850" />
        <polyplane Rio_Z="5.484013 7.957603 53.159908" />
        <polyplane Rio_Z="1.667943 4.141533 58.609820" />
        <polyplane Rio_Z="3.327747 3.327747 59.772027" />
</pgon>
```

```
vector<vector<double> > sc_rioz;
dgeom->GetMultiple("//StartCntr_s/section/pgon/polyplane/@Rio_Z", sc_rioz);
```

Add log function to *JObject*

```
void AddLog(string &message) const
void AddLog(vector<string> &messages) const
void GetLog(vector<string> &messagelog) const
```

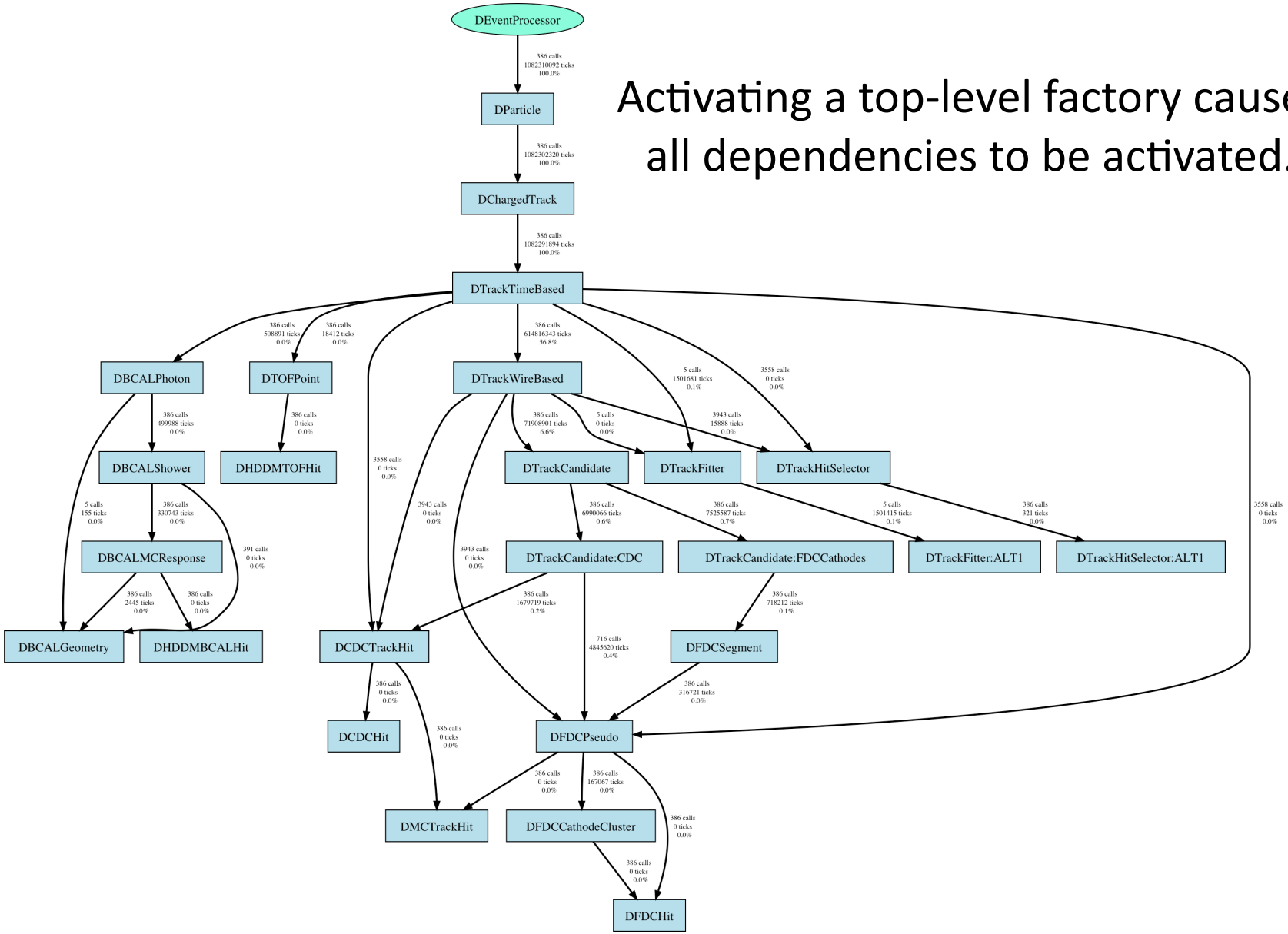
Allow log messages to be attached to single objects. (Could be useful in debug mode.)

Message log is mutable so factories can add to it using const. pointers

janaroot plugin honors *WRITEOUT* configuration parameter

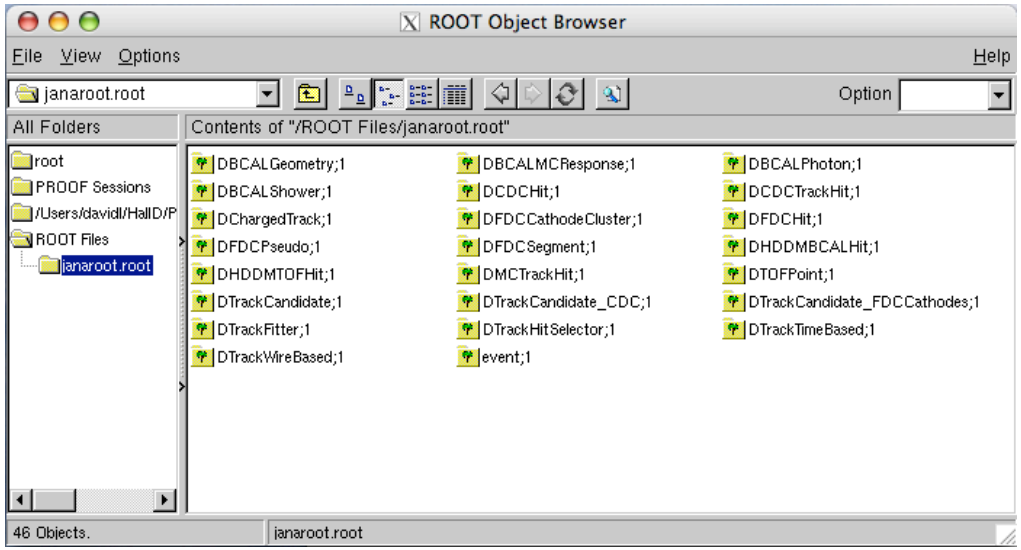
- By default, *janaroot* writes out objects for every object created during event reconstruction.
- If the *WRITEOUT* config. parameter is set, it will specify the objects to write out
- Un-tagged factories will be specified by the name of the data object. Tagged factories will append a ":Tag"

```
>hd_ana -PPLUGINS=janaroot,janadot --auto_activate=DParticle hdgeant_smeared.hddm
```



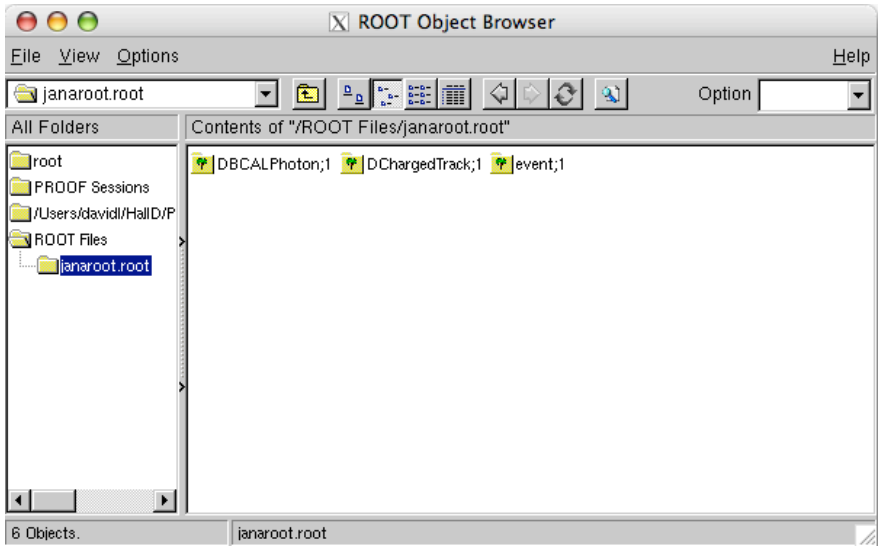
Activating a top-level factory causes all dependencies to be activated.

janaroot auto-magically creates *TTrees*




```
hd_ana -PPLUGINS=janaroot,janadot
 --auto_activate=DParticle
hdgeant_smeared.hddm
```

WRITEOUT config. parameter not specified. All JANA objects made into trees




```
hd_ana -PPLUGINS=janaroot,janadot
-PWRITEOUT=DChargedTrack,DBCALPhoton
--auto_activate=Dparticle
hdgeant_smeared.hddm
```

WRITEOUT config. parameter such that only DChargedTrack and DBCALPhoton objects are written out as trees

Monitoring and Control with janactl

- *janactl* is the name used for both the plugin and a command-line utility
- Any JANA program can attach the *janactl* plugin to allow external monitoring and control
  - Data Quality monitoring
  - Level 3 trigger
  - Offline farm jobs
  - Local jobs (e.g. processing simulated data)

Currently Implemented Commands

Commands *janactl* plugin understands

- *"who's there?"*
- *"get threads"*
- *"pause"*
- *"resume"*
- *"quit"*
- *"kill"*

Replies *janactl* plugin sends

- *"I am here"*
- *"thread info"*

All communications are done using a passive model (i.e. *send and forget*).

Thread info. via *janactl*

This output shows the result of a "*get threads*" command issued when 2 processes were listening.

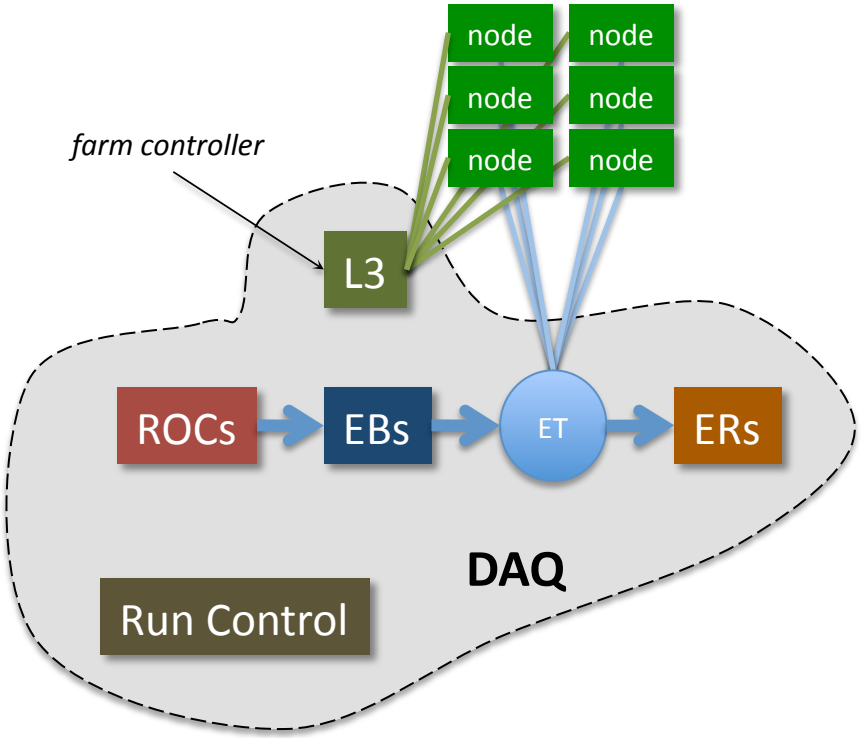
One had a single processing thread while the other had three processing threads

```
~>janactl thinfo -t 0.1
parseUDL: udl remainder = localhost/cMsg/janactl
parseUDL: host = localhost
parseUDL: host = Amelia.local
parseUDL: mustMulticast = 0
parseUDL: TCP port = 45000
parseUDL: subdomain = cMsg
parseUDL: subdomain remainder = janactl, len = 7
DONE PARSING UDL
Sent command: get threads to janactl

Threads by process:
-------------------------------------------------
Amelia.local_29219:
   thr. 0xb0491000  109 events  2.63513Hz (4.33184Hz avg.)
Amelia.local_29220:
   thr. 0xb0491000  42 events  3.29637Hz (1.98569Hz avg.)
   thr. 0xb0513000  42 events  3.32588Hz (2.00329Hz avg.)
   thr. 0xb0595000  33 events  1.63243Hz (1.57239Hz avg.)
```

```
Usage:
       janactl [options] cmd

Communicate with remote or local processes that have the janactl
plugin attached.

Options:

   -h, --help  Print this message
   -t timeout  Set the timeout of commands while waiting for a response.
   -u udl      Set UDL of cMsg server. If not given, the JANACTL_UDL environment
               variable is used. If that's not set, the localhost is used.
   -d descr.   Set description text of this program for use by cMsg server.
   -s subject  Send command to subject (default is all "janactl" which is
               all processes.

   --timeout timeout      Same as -t
   --udl udl              Same as -u
   --name name            Same as -n
   --description descr.   Same as -d
   --subject subject      Same as -s

The janactl system uses a passive communication mechanism for control. Specifically,
messages are sent, but responses (if any) are received asynchronously. As such
commands such as the "list" command simply broadcasts a query to all processes
that asks for them to send back a message notifying us of their existence. At
some point, we must decide all responses have been received and continue on.
the "timeout" for this is set by default to 3 seconds, but an alternate may be set
via the "-t timeout" command line switch. Commands that expect only a single
response will continue as soon as that response is received.

 commands:
   list        Lists available processes.
   pause       Pause remote process(es)
   resume      Resume remote process(es)
   quit        Quit remote process(es) gracefully
   kill        Kill remote process(es) harshly
   thinfo      Get thread info. for remote process(es)
```

Integration with DAQ system

- A L3 farm will be a component of the DAQ system
- The farm is a nebulous entity in that nodes/processes may come and go during normal operation
- The proposed model would use a *farm controller* program that would present the farm to the DAQ as a single *codaobject*



Other misc.

- NTHREADS config. Parameter specifies number of processing threads (can be set to *Ncores* to feel out number of cores on current system)
- Added *GetSingle()* method to *JObject*
- *Compiled (but not vigorously tested on)*
  - *Linux_CentOS5-x86_64-gcc4.1.2*
  - *Linux_RHEL5-i686-gcc4.1.2*
  - *Linux_Fedora8-i686-gcc4.1.2*
  - *Darwin_macosx10.5-i386-gcc4.0.1*
  - *Darwin_macosx10.6-x86_64-gcc4.2.1*