Recent Developments in *JANA*

David Lawrence JLab

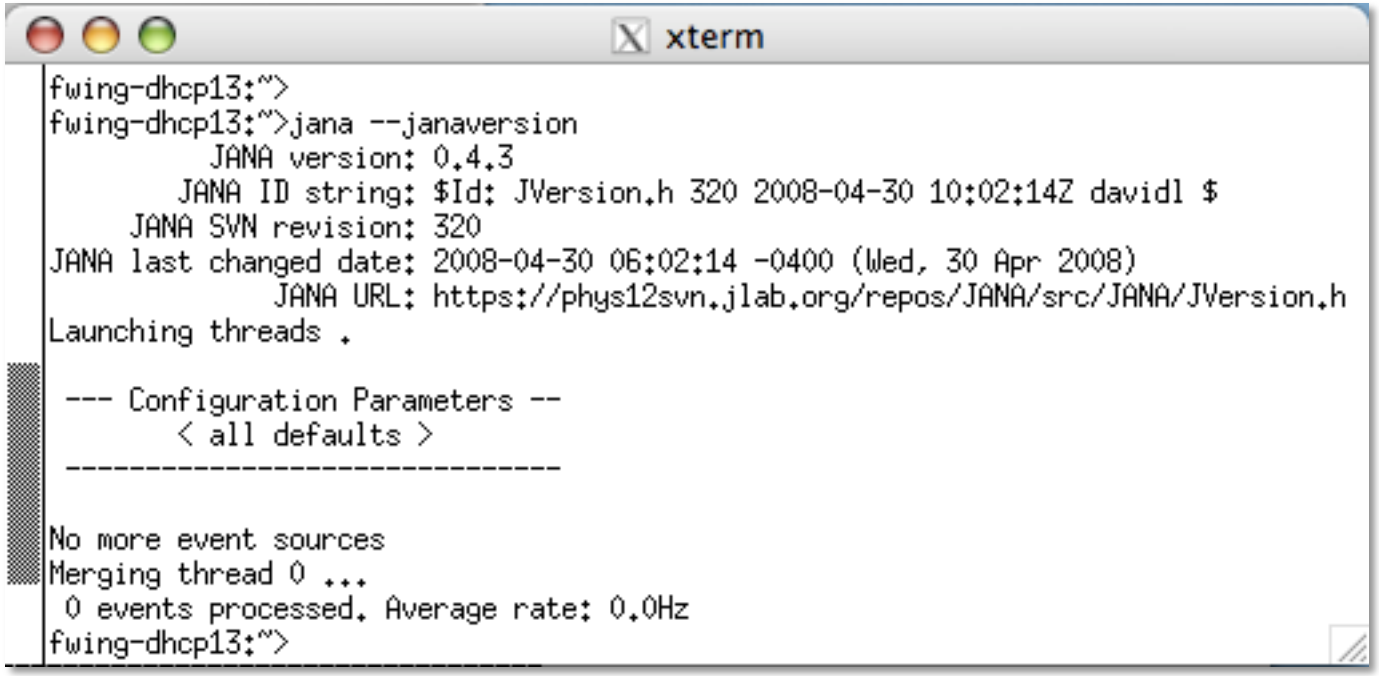
May 23, 2008

JANA has been placed inside of the "jana" namespace

- Better isolates JANA when mixing with other packages
- Removed lots of:
  "`using namespace std;`"
  lines from headers
- Added lots of:
  "`using namespace jana;`"
  lines to .cc files

Added *JVersion.h*

The *JVersion.h* file helps keeps track of the version of jana used in the current program. Now, all *JANA* programs can display their jana version via the --*janaversion* command line argument.



JANA build system now uses *autoconf*

- When building *JANA* from source, one now starts with a ***configure*** script that is included in the src directory.
- ***configure*** will create the files:
  BMS/jana_conf.mk
  JANA/jana_config.h
- *Currently, it is used to sense Xerces and MySQL*

JANA build system now uses *autoconf*

Using *autoconf* automatically provides the standard interface for *configure* scripts.

```
fwing-dhcp13:src>./configure --help
`configure' configures janaconfig version-0.1 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...
...
Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
  --with-mysql=[ARG]      use MySQL client library [default=yes], optionally
                          specify path to mysql_config
  --with-xerces=[ARG]     use Xerces C++ Parser from given prefix (ARG=path);
                          check standard prefixes (ARG=yes); disable (ARG=no)
  --with-xerces-inc=[DIR] path to Xerces C++ Parser headers
  --with-xerces-lib=[ARG] link options for Xerces C++ Parser libraries
...
```

toString() method replaced by *toStrings()*

Previously, programs like *hd_dump* would print data objects using the *toString()* method of the factory class that created them. This was less than ideal because:

- Multiple factories that produced the same type of data objects had to duplicate the *toString* method
- Data objects originating at the source (e.g. *DMCThrown*) had to have an associated factory class just to implement *toString*

Example of original *toString* method in JANA

```
25  const string DMCThrown_factory::toString(void)
26  {
27     // Ensure our Get method has been called so _data is up to date
28     GetNrows();
29     if(_data.size()<=0)return string(); // don't print anything if we have no data!
30
31     printheader("      id: type:  q:     p:    E: theta:   phi:   mass:     x:     y:     z:");
32
33     for(unsigned int i=0; i<_data.size(); i++){
34        DMCThrown * mcthrown = _data[i];
35
36        printnewrow();
37
38        printcol("%lx", mcthrown->id);
39        printcol("%d", mcthrown->type);
40        printcol("%+d", (int)mcthrown->q);
41        printcol("%3.3f", mcthrown->p);
42        printcol("%3.1f", mcthrown->E);
43        printcol("%1.3f", mcthrown->theta);
44        printcol("%1.3f", mcthrown->phi);
45        printcol("%1.3f", mcthrown->mass);
46        printcol("%2.2f", mcthrown->x);
47        printcol("%2.2f", mcthrown->y);
48        printcol("%2.2f", mcthrown->z);
49
50        printrow();
51     }
52
53     return _table;
54  }
```

Example of new *toStrings* method in JANA

```
16  class DMCThrown:public DKinematicData{
17     public:
18        JOBJECT_PUBLIC(DMCThrown);
19
20        int type;          ///< GEANT particle ID
21        int pdgtype;       ///< PDG particle type (not used by GEANT)
22        int myid;          ///< id of this particle from original generator
23        int parentid;      ///< id of parent of this particle from original generator
24        int mech;          ///< production mechanism of this partcle (generator specific)
25
26        void toStrings(vector<pair<string,string> > &items)const{
27           AddString(items, "type", "%d", type);
28           AddString(items, "pdgtype", "%d", pdgtype);
29           AddString(items, "myid", "%d", myid);
30           AddString(items, "parentid", "%d", parentid);
31           AddString(items, "mech", "%d", mech);
32           DKinematicData::toStrings(items);
33        }
34
35  };
```

Source-present objects still need a factory, but not a sub-class of *JFactory*

Translation: at least 16 .h and .cc files were removed due to this simplification

```
3  #include <JANA/JEventLoop.h>
4  #include "DBCALMCResponse_factory.h"
5  #include "DBCALGeometry_factory.h"
6  #include "DBCALShower_factory.h"
7  #include "DBCALShower_factory_IU.h"
8  #include "DBCALPhoton_factory.h"
9
10 #include "DBCALHit.h"
11 #include "DBCALTruthShower.h"
12 #include "DHDDMBCALHit.h"
13 typedef JFactory<DBCALHit> DBCALHit_factory;
14 typedef JFactory<DBCALTruthShower> DBCALTruthShower_factory;
15 typedef JFactory<DHDDMBCALHit> DMCBCALHit_factory;
16
17 jerror_t BCAL_init(JEventLoop *loop)
18 {
19    /// Create and register BCAL data factories
20    loop->AddFactory(new DBCALHit_factory());
21    loop->AddFactory(new DBCALMCResponse_factory());
22    loop->AddFactory(new DBCALGeometry_factory());
23    loop->AddFactory(new DBCALShower_factory());
24    loop->AddFactory(new DBCALShower_factory_IU());
25    loop->AddFactory(new DBCALTruthShower_factory());
26    loop->AddFactory(new DBCALPhoton_factory());
27    loop->AddFactory(new DMCBCALHit_factory());
28
29    return NOERROR;
30 }
```

Associated *JObjects*

With JANA 0.4.4, an *associated objects* list was added to the *JObject* class. The primary purpose is to allow one to reference the input data objects used to create an ouput data object. Multiple types can be stored as long as they all inherit from *JObject*.

Example uses:

- Record the hits used to form a shower
- Record candidate associated with a fit track

Associated Objects Example

```
27    vector<const DBCALMCResponse*> myhits;
28
29    // ... find shower and fill myhits with hits ...
30
31    DBCALShower *shower = new DBCALShower;
32    for(unsigned int i=0; i<myhits.size(); i++)
33       shower->AddAssociatedObject(myhits[i]);
34    }
35    shower->AddAssociatedObject(bcalGeom);
```

Storing the *DBCALMCResponse* objects as associated objects to a *DBCALShower* object

```
37    vector<const DBCALShower*> showers;
38    loop->Get(showers);
39
40    for(unsigned int i=0; i<showers.size(); i++){
41       vector<const DBCALMCResponse*> myhits;
42       showers[i]->Get(myhits);
43    }
```

Retrieving the *DBCALMCResponse* objects from the *DBCALShower* object

JGeometry

- Facilities now exist in *JANA* to help manage and access detector geometry information.
- The *JGeometry* base class presents an API that is agnostic as to the underlying storage or retrieval mechanism (e.g. is the information stored in local XML files or in a MySQL database on the network?)
- A generic utility *jgeomread* can be used to access the geometry from the command line.

Summary

- *JANA* now lives in the “jana” namespace
- Version information is now compiled into every JANA program and is accessible on the command line using the *–janaversion* switch
- *JANA* now uses *autoconf* in its standard build procedure
- *toString()* methods in factories replaced by t*oStrings()* methods in *JObject*
- Associated objects now in *JObject*
- *JGeometry* class and *jgeomread* utility added to *JANA*