Analysis Group DDL Managment

- Test Release
  - One Time Work
    - Make a conda based test release
    - Activate it
    - Add Packages
    - pdsdata note
    - Build, make sure it works
    - More Packages?
- Flow
  - Clean
  - New Tags
  - Create Generated Code
    - High Priority Packages
    - Low Priority: psana_test
    - Low Priority: Hdf5
    - Notes
  - Build and Test
- Code Links

The DAQ will produce a new pdsdata and psddldata per the instructions here: Building the psalg and pdsdata packages

Test Release

We need to setup a test release for DDL development.

One Time Work

Make a conda based test release

```
source conda_setup
condarel --newrel --names types-conda
```

Activate it

```
cd types-conda
source conda_setup
```

you should get the *tr* in your prompt

Add Packages

Right now your types-conda is against ana-current, ana-1.2.7 for me. Packages we expect to modify, we'll check out against master/head. Otherwise we'll let condarel get the package tag for our release.

The next time we update a DDL type and we re-use this release, we may have to update tags.

```
condarel --addpkg --name pdsdata --tag HEAD
condarel --addpkg --name psddldata --tag HEAD

# we need this proxy to use pdsdata, shouldn't need to get it from master
condarel --addpkg --name pdsdata_ext

# we'll need these proxy packages for compiling the Translator
condarel --addpkg --name hdf5
condarel --addpkg --name openmpi

# we'll  need this proxy package to compile psddl_python
condarel --addpkg --name python

# These are the packages that we will generate new code for, so we'll get them from head

# these are essential:
condarel --addpkg --name psddl_psana --tag HEAD
condarel --addpkg --name psddl_pds2psana --tag HEAD
condarel --addpkg --name psddl_python --tag HEAD


# psana_test is part of the testing system, and useful for dummping xtc files, but could be deprecated,
probably no user uses it
condarel --addpkg --name psana_test --tag HEAD

# this if for the Translator
condarel --addpkg --name psddl_hdf2psana --tag HEAD
condarel --addpkg --name Translator --tag HEAD
```

pdsdata note

The code for pdsdata is checked out into the extpkgs subdir, if you need to modify or run svn log or things like that, cd into the appropriate directory

Build, make sure it works

scons

or better,

scons test

More Packages?

Above should be the essential, but it might be a good idea to include more packages, for instance all the packages that depend on these packages, and psana. Below is the list of all the packages I kept in my RPM based types release:

```
pslogin7d: /reg/neh/home4/davidsch/rel/types $ relinfo
Pkg/Rel           TEST        BASE(ana-0.19.25)
CSPadPixCoords    V00-03-30   V00-03-30           V00-03-30
CalibManager      V00-02-26   V00-02-26           V00-02-26
CythonUtils       V00-00-01   V00-00-01           V00-00-01
Detector          V00-04-13   V00-04-13           V00-04-13
H5DataTypes       V00-05-14   V00-05-14           V00-05-14
ImgAlgos          V00-04-37   V00-04-37           V00-04-37
LogBook           V01-01-05   V01-01-05           V01-01-05
LusiTime          V00-01-08   V00-01-08           V00-01-08
MsgLogger         V00-01-09   V00-01-09           V00-01-09
PSCalib           V00-03-20   V00-03-20           V00-03-20
PSEnv             V00-14-01   V00-14-01           V00-14-01
PSEvt             V00-08-05   V00-08-05           V00-08-05
PSHdf5Input       V00-04-02   V00-04-02           V00-04-02
PSShmemInput      V00-01-02   V00-01-02           V00-01-02
PSXtcInput        V00-15-22   V00-15-22           V00-15-22
PSXtcMPInput      V00-00-02   V00-00-02           V00-00-02
ParCorAna         V00-00-35   V00-00-35           V00-00-35
TimeTool          V00-03-11   V00-03-11           V00-03-11
Translator        HEAD        HEAD                V00-02-61
XtcInput          V00-10-28   V00-10-28           V00-10-28
boost             V00-03-00   V00-03-00           V00-03-00
cspad_mod         V00-07-00   V00-07-00           V00-07-00
doctools          V00-00-22   V00-00-22           V00-00-22
h5py              V00-06-01   V00-06-01           V00-06-01
hdf5              V00-05-01   V00-05-01           V00-05-01
hdf5pp            V00-07-09   V00-07-09           V00-07-09
hypre             V00-00-01   V00-00-01           V00-00-01
mpi4py            V00-01-01   V00-01-01           V00-01-01
mypkg             -           -
mysql             V00-02-01   V00-02-01           V00-02-01
openmpi           V00-02-00   V00-02-00           V00-02-00
pdsdata           HEAD        HEAD                V01-00-51
psalg             V00-00-07   V00-00-07           V00-00-07
psana             V00-13-24   V00-13-24           V00-13-24
psana_examples    V00-05-03   V00-05-03           V00-05-03
psana_python      V00-15-21   V00-15-21           V00-15-21
psana_test        HEAD        HEAD                V00-08-57
psddl             V00-13-25   V00-13-25           V00-13-25
psddl_hdf2psana   HEAD        HEAD                V00-09-40
psddl_pds2psana   HEAD        HEAD                V00-09-36
psddl_psana       HEAD        HEAD                V00-06-31
psddl_python      HEAD        HEAD                V00-08-30
psddldata         V01-00-53   V01-00-53           V01-00-53
psocake           V00-01-52   V00-01-52           V00-01-52
pyana             V00-03-05   V00-03-05           V00-03-05
pyana_examples    V00-01-02   V00-01-02           V00-01-02
pyimgalgos        V00-00-74   V00-00-74           V00-00-74
pypsalg           V00-01-13   V00-01-13           V00-01-13
pytools           V00-00-02   V00-00-02           V00-00-02
tables            V00-06-01   V00-06-01           V00-06-01
```

one can also use

`scons package-revese-dependencies`

to identify what packages depend on the psddl packages, including those would let you develop changes that brake binary compatibility.

Flow

Clean

We're going to want to rebuild everyting against the new tags, I suggest, from your types-conda directory

`rm -rf build arch include data`

you can also do

```
scons clean
```

but not as reliable

New Tags

get the new tags from the DAQ for pdsdata and psddldata, I would use git checkout for the later so you still have your whole git checkout (instead of a headless branch)

```
# this from the test release dir, changes code in the extpkgs/pdsdata subdir:
condarel --addpkg --name pdsdata --tag V08-07-00
cd psddldata
git pull
# say you want this tag:
git checkout V01-00-53
```

The DAQ will have checked in a pdsdata with code generated from the new DDL type in psddldata. Now we will generate our own code from this new type in psddldata.

Create Generated Code

High Priority Packages

To create generated code, we will run the driver scripts for each of the packages. That is we will run, in this order (but the order doesn't really matter)

```
ddl_psana           # build abstract C++ interfaces to new DDL type in psddl_psana package
ddl_pds2psana       # build C++ derived interface that uses pdsdata to understand xtc files, in psddl_pds2psana
package
ddl_python          # build python extension, in psddl_python package
```

These are the essential ones for python psana users.

It can be useful to go into the three psddl_* packages mentioned and do git status to see differences.

You could run scons now and try out psana and see if you see the new DDL type. If you have a test xtc file, you could take a look at it (it is great if you get a test xtc file before running live in experiments)

Low Priority: psana_test

Next, or during, also run this code generator:

```
ddl_psana_test      # extend the psana_test.dump psana module to understand the new datatype, now you can get a
text dump of this new DDL type
```

Low Priority: Hdf5

Before generating code for the Translator, you usually need to write a litttle DDL. This DDL describes how to write the new DDL type into Hdf5 - basically how many datasets do we split the data into, and how do we group it, do we use compounds, etc. Best to look at examples of past commits to the psddl_hdf2psana package. For example, here is what was added for the juafrau detctor, based on this DDL

One is using the DDL to write a schema. Schema's get there own versions. For example, today, we may write DDL type V3 with schema 1, but if one fines one must change the names of the datasets or something like that, you can introduce a schema 2 without changing the DDL.

```
ddl_hdf2psana      # C++ classes to read write details of DDL types to/from Hdf5 per the schemas in the
psddl_hdf2psana/data directory.
ddl_Translator     # C++ for the Translator, uses ddl_hdf2psana for low level type I/O
```

Notes

We no longer use/maintain ddl_psanadoc.

All the ddl_* scripts call the underlying psddlc tool in the psddl package. They are all pythyon scripts that take -h, for instance they have a verbose switch, and for debugging, you can run them on one file at a time.

When you run them, you get some messages you can ignore, for instance:

```
(ana-1.2.7) *tr* psanaphi107: ~/rel/types-conda $ ddl_psana
Warning: <Package(Epix)> type=ElementV2, DEVEL type=ConfigSV1 in config list is being omitted
Warning: <Package(Epix)> type=ElementV3, DEVEL type=ConfigSV1 in config list is being omitted
```

At some point we decided to introduce a DEVEL tag. The idea being that the DAQ would mark volatile types in development as DEVEL which we would not generate interfaces for. This way users might not accidentally use old devel code for new production types (once the type was stable). It does create issues, this is just warning you that a DEVEL config type is listed in a production data type, but we're ignoring it (maybe this message should be removed, as it is the obvious thing to do).

```
(ana-1.2.7) *tr* psanaphi107: ~/rel/types-conda $ ddl_pds2psana
WARNING No suitable constructor defined for Acqiris::TdcDataV1Common
WARNING No suitable constructor defined for Acqiris::TdcDataV1Channel
WARNING No suitable constructor defined for Acqiris::TdcDataV1Marker
```

Those have been there forever, no worries.

```
DdlPythonInterfaces - info: BldDataFEEGasDetEnergy is a type family including
  names ending with and without the version string.
  Not generating the unversioned object containing the versioned types.
```

In our psana package, you'll see things like

```
psana.BldEBeam
psana.BldEBeamV1
psana.BldEBeamV2
```

where BldEBeam is a list of the other types. It is a way to programmatically figure out what types are in a release. However for an old type like FEEGas - they didn't start with a Vx, so we can't make this list. Just a warning as we see there is more than on version of the type, and we'd like to make a list but can't.

```
(ana-1.2.7) *tr* psanaphi107: ~/rel/types-conda $ ddl_psana_test
notProcessed 111: uint8_t[1] <- Generic1D.DataV0.data_u8( [('channel', <Type(uint32_t)>)] ) value_type=True
notProcessed 110: double <- OceanOptics.DataV1.nonlinerCorrected( [('iPixel', <Type(uint32_t)>)] )
value_type=True
...
```

I never implemented the code to dump more complicated methods of the DDL types - for instance nonlinearCorrected is a function that takes an index, we'd have to call it repeatedly.

Build and Test

now the fun:

```
scons test
```

good luck!

Code Links

https://github.com/lcls-psana/psddl_hdf2psana/commits

https://github.com/lcls-psana/psddldata/commits

https://github.com/lcls-psana/psddl/commits

https://github.com/lcls-psana/psddl_psana/commits

https://github.com/lcls-psana/psddl_pds2psana/commits

https://github.com/lcls-psana/psddl_python/commits

https://github.com/lcls-psana/psana_test/commits