Outline

- What is a dictionary
- What is a proxy
- Why is the existing proxy dict useful for the Panda offline software
  - general characteristics
  - for the event loop
  - for time dependent objects
- Examples
  - list of references to objects with event-live-time
  - associative maps with event-live-time

Dictionary

Definition: Data dictionary (from web)

A data dictionary is a collection of descriptions of data objects or items in the data model for the benefit of programmers or others who need to refer to them.
A first step in analyzing a system of objects with which user interacts is to identify each object and its relationship to other objects. This process is called data modeling and results in a picture of object relationships.
...
This collection can be organized for reference into a book called data dictionary.

Dictionary

Definition (in my words for Panda purposes)

A dictionary is a central place where objects are collected.
The storage of and the access to these objects can be done in each module/class that has access to the dictionary.
In general, the data are associated with a key (e.g. string, time interval etc.) which makes the identification and the access to the objects very easy.

Use Cases in PandaRoot

- facilitate sharing of objects (between different tasks)
  - ensures that there is a single authoritative source of reference for all users/clients
- simplify code development
  - a simple API to access event data

Proxy

Well known: Web browsers make use of the proxy mechanism

Def: Proxy server

A server that sits between a client application, such as a web browser, and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.
Example: Caching of web pages

Benefits of the proxy mechanism

- dramatic improvement of the performance
  - it saves (caches) the answers of all requests for a certain amount of time

Existing Proxy Dict for PANDA

- General characteristics
  - type-save
    - casts are not needed in the users code
    - allows OO code design
    - recognizing bugs at compile time
  - supports std::vector containing object references which are deleted automatically
  - supports all kinds of objects, i.e. even the association objects

Existing Proxy Dict for PANDA

- Event based proxy dict
  - collects event based objects
  - make it easy to share information between different tasks
    - objects can be stored in one task and retrieved by all following tasks in e.g. the Exec() function

- Job live-time proxy dict
  - useful to collect conditions objects
  - API to the CDB
    - improves the performance dramatically due to the proxy mechanism
    - user/client has not to take care of updating the objects
    - user/client just gets the valid objects

Example: Event-Live-Time Object References

Task A: stores a list of object references into the dictionary

```
#include “Ifd.h”
#include “IfdStdVectorList.h”
#include “AbsEvt.h”
...
TaskA::Exec(){
 std::vector <PndEmcWaveform*> * theWaveList
                                        = new std::vector <PndEmcWaveform*>

IfdStdVectorProxy<PndEmcWaveform*>  *stdProxyWaveform
                 = new IfdStdVectorProxy<PndEmcWaveform*> ( theWaveformList  );

Ifd< vector<PndEmcWaveform*> >::put( gblEvtDict, stdProxyWaveform );

...
for (Int_t iHit=0; iHit<nHits; iHit++) {
              theHit = (PndEmcHit*) fHitArray->At(iHit);
             theWaveformList->push_back(new PndEmcWaveform);
  }

}
```

event based dictionary

put it into the dictionary

filling the std::vector

Example: Event-Live-Time Object references

Task B: retrieve the list of object references from the dictionary

```
#include “Ifd.h”
#include “IfdStdVectorList.h”
#include “AbsEvt.h”
...
TaskB::Exec(){

std::vector<PndEmcWaveform*> *waveformList
                  = Ifd< std::vector<PndEmcWaveform*> >::get(gblEvtDict);

if (0==waveformList) Fatal(“TaskB”,”retrieve 0 pointer from dict”);
...
std::vector<PndEmcWaveform*>::const_iterator it=waveformList->begin();
        for (it; it!=waveformList->end();++it)
          {
           . . .
          }
}
```

std::vector
from proxy dict

Example: Event-Live-Time Maps

Task A: stores a map into the proxy dict

```
#include "AstSTLMap2.h"
...
...
TaskA::Exec(){
 ...
  typedef AstSTLMap2<PndEmcHit, PndEmcWaveform> EmcHitWaveMap;
  EmcHitWaveMap * emcHitWaveMap1 = new EmcHitWaveMap;
  EmcHitWaveMap * emcHitWaveMap2 = new EmcHitWaveMap;
  Ifd<EmcHitWaveMap>::put(gblEvtDict
                        , new IfdDataProxy<EmcHitWaveMap> (emcHitWaveMap1),"Default");
  Ifd<EmcHitWaveMap>::put(gblEvtDict
                        , new IfdDataProxy<EmcHitWaveMap> (emcHitWaveMap2),"Ecut");
...
for (Int_t iHit=0; iHit<nHits; iHit++) {
              theHit = (PndEmcHit*) fHitArray->At(iHit);
              PndWaveform* theWaveform = new PndWaveform(...);
              theWaveformList->push_back(theWaveform);

              emcHitWaveMap1->append( theHit, theWaveform);
              if (theHit->GetEnergy()>4.0){
                                emcHitWaveMap2->append( theHit, theWaveform);
               }
  }
}
```

Maps with different keys

Storage of one element in map with key “Default”

Storage of one element in map with key “Ecut”

Example: Event-Live-Time Maps

Task B: retrieve the map from the proxy dict

```
#include “AstSTLMap2.h”
...
TaskB::Exec(){
...
 typedef AstSTLMap2<PndEmcHit, PndEmcWaveform> EmcHitWaveMap;
 EmcHitWaveMap * emcHitWaveMap
  = Ifd<AstSTLMap2<PndEmcHit, PndEmcWaveform> >::get(gblEvtDict,"Default");

if (0==emcHitWaveMap)  Fatal(“TaskB”,”retrieve 0 pointer from dictionary”);
...
std::vector<PndEmcWaveform*>::const_iterator it=waveformList->begin();
        for (it; it!=waveformList->end();++it)
          {
           ...
            PndEmcHit* theMatchedHit=emcHitWaveMap->findFirstValue2((*it));
            if (0==theMatchedHit) Fatal(“TaskB”,”theMatchedHit==0”);
          }
}
```

map with key “Default” from proxy dict

first associated EmcHit to the Waveform (*it)

Summary

- Definition of a dictionary
- Definition of a proxy
- Existing proxy dict is an effective tool to store and retrieve (transient) objects at a central place
  - makes it easy to share (transient) objects between different tasks
- Existing proxy dict can be used as an interface to the CBD
  - improvement of the performance