Configuration Concept

S. Pflüger

Helmholtz Institute Mainz

September 8, 2015

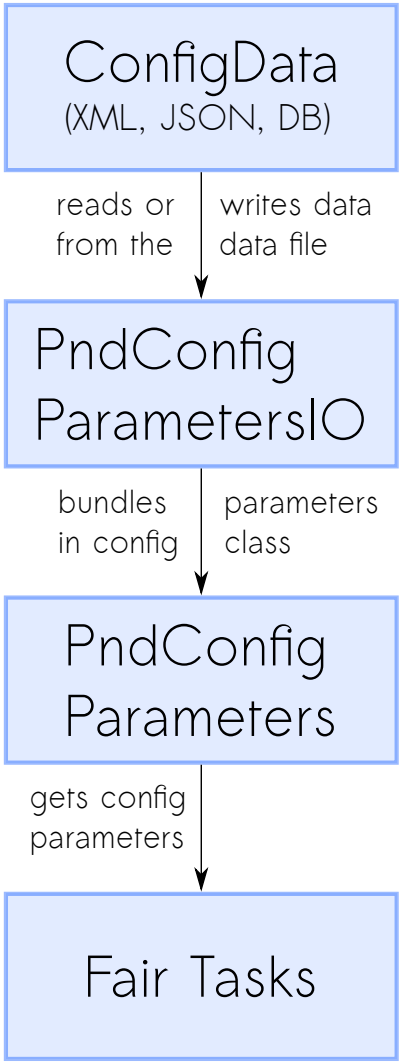
What started this?

Lmd Macro calls

```
# #digi
     root -l -b -q runLumiPixel1Digi.C\(${numEv},0,"\"${path}\"",0\)

# #hit reconstruction
     root -l -b -q runLumiPixel2Reco.C\(${numEv},0,"\"${path}\"",0,false\)

# ##merge hits
     root -l -b -q runLumiPixel2bHitMerge.C\(${numEv},0,"\"${path}\""\)

### change "CA" --> "Follow" if you want to use Trk-Following as trk-search algorithm
### NB: CA can use merged or single(not merged) hits, Trk-Following can't
#      root -l -b -q runLumiPixel3Finder.C\(${numEv},0,"\"${path}\"",0,"\"CA\"",${misspl},${mergedHits},${trkcut},${pbeam}\)
     root -l -b -q runLumiPixel3Finder.C\(${numEv},0,"\"${path}\"",0,"\"Follow\"",${misspl},${mergedHits},${trkcut},${pbeam}\)
#track fit:
### Possible options: "Minuit", "KalmanGeane", "KalmanRK"
### radLen = average effective thickness X/X0[%]
       radLen=0.32
       root -l -b -q runLumiPixel4Fitter.C\(${numEv},0,"\"${path}\"",0,"\"Minuit\"",${mergedHits},${radLen}\)

#track filter (on number of hits and chi2)
     root -l -b -q runLumiPixel4aFilter.C\(${numEv},0,"\"${path}\"",0,${mergedHits},${SkipFilt},${XThetaCut},${YPhiCut},${BoxCut},${dX},${dY}\)

#save filtered results as standart Track array
     mv ${path}/Lumi_Track_0.root ${path}/Lumi_TrackNotFiltered_0.root
     cp ${path}/Lumi_TrackFiltered_0.root ${path}/Lumi_Track_0.root

#back-propgation GEANE
### Possible options: "Geane", "RK"
     root -l -b -q runLumiPixel5BackProp.C\(${numEv},0,"\"${path}\"",0,"\"Geane\"",${mergedHits},${pbeam}\)
```

- many macro calls with large number of parameters

My Concept

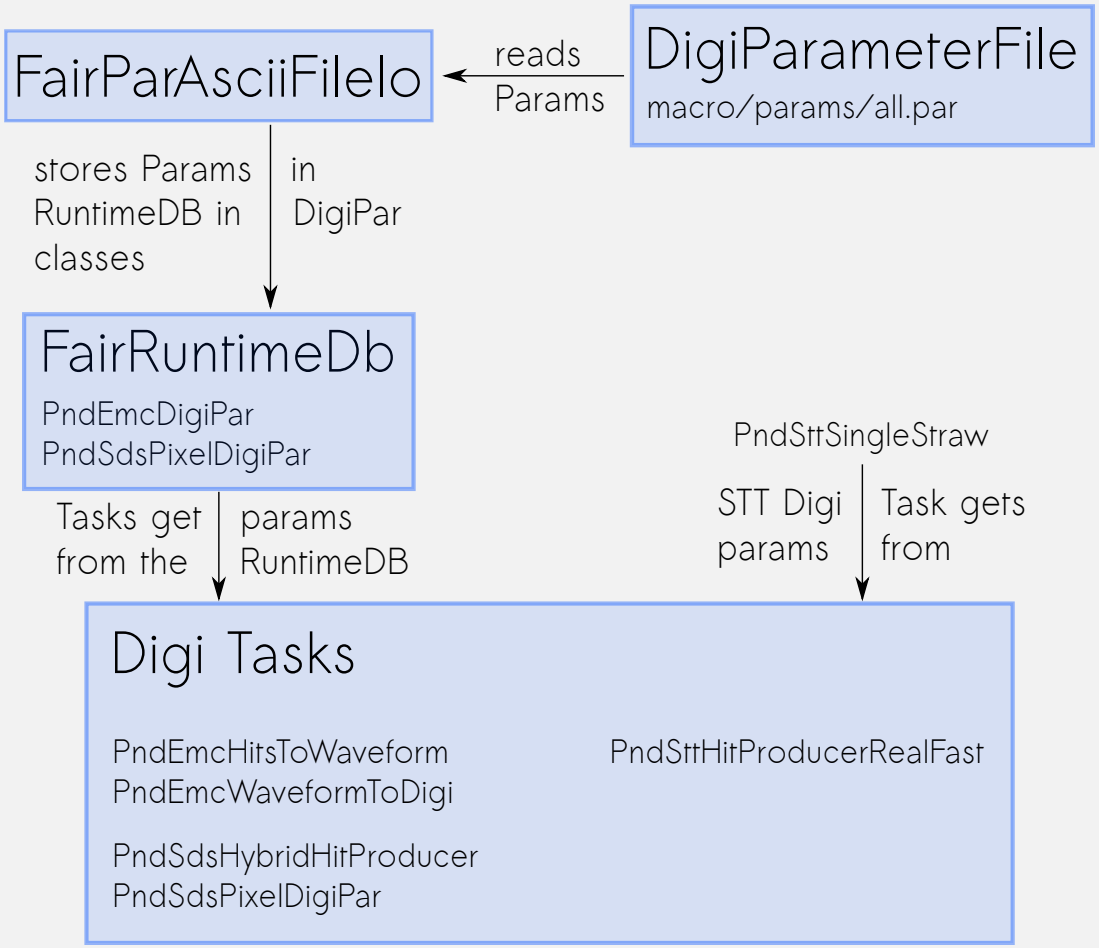



```
void testmacro(std::string config_url) {
  gSystem->Load("../lib/libPndConfigParameters.so");
  //gSystem->SetIncludePath("");

  PndConfigParametersIO reader;

  PndConfigParameters sim_conf = reader.readConfigFromFile(config_url);

  cout<<"num events: "<<sim_conf.getParameterValueByName<int>("sim.num_events")<<endl;
  cout<<"use_y_phi_cut: "<<sim_conf.getParameterValueByName<bool>("reco.use_y_phi_cut")<<endl;
  cout<<"ip_offset_x: "<<sim_conf.getParameterValueByName<bool>("ip_beam.ip_mean_x")<<endl;
}
```

- simple, extensible and DRY concept
- config parameters have to be accessed by name and access errors only available at runtime

How its done in PANDAROOT?

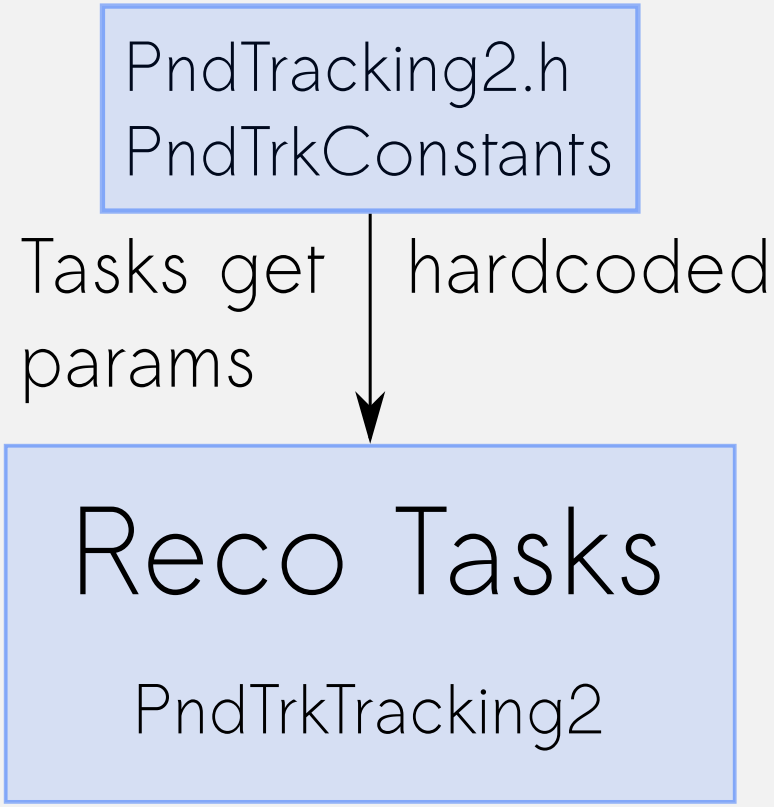
Digi Case (digi_complete.C)



- design is all in all fine
- However:
  - each parameter group requires "special" class as container (e.g. PndEmcDigiPar)
  - changes in the parameter files require changes in these classes

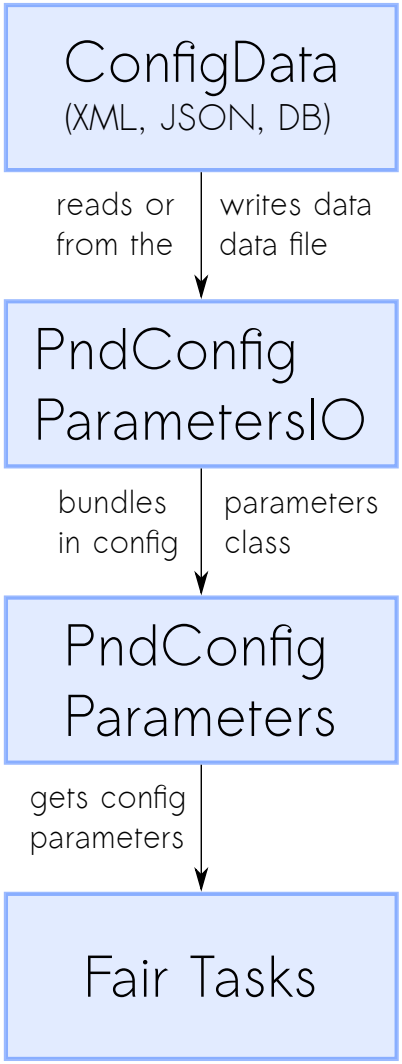
How its done in PANDAROOT?

Reco Case (reco_complete.C)



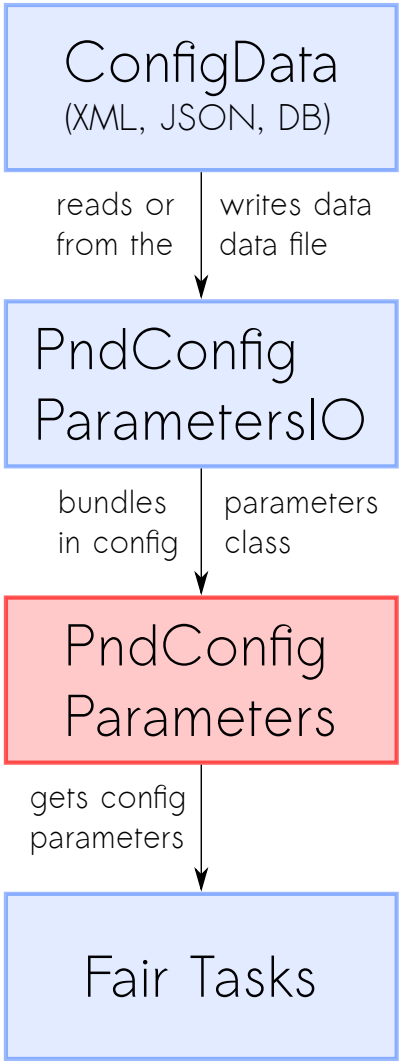
- parameters are hardcoded in serveral classes
- why?

My Concept



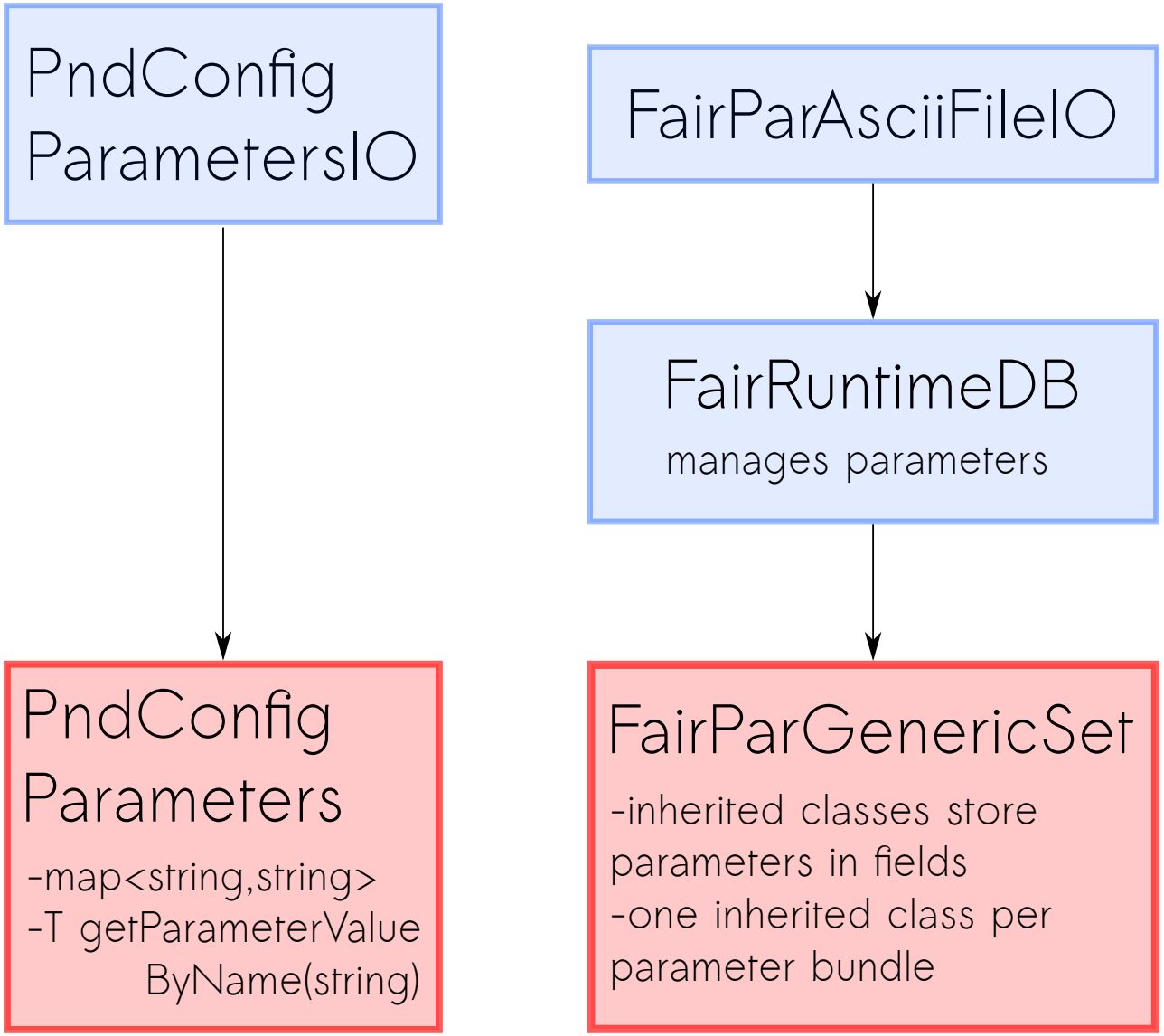
- pretty similar to forseen workflow via runtime DB
- main difference: do not store parameters in "special" classes but in general map container
- advantage: no special classes are needed

My Concept



- pretty similar to forseen workflow via runtime DB
- main difference: do not store parameters in "special" classes but in general map container
- advantage: no special classes are needed

Concept Difference



Summary

- digi task looks fine (is stt conform?)
- reco task looks different, why?
- fairroot design requires container classes for each parameter "bundle"

Conclusions

- possible change to fairroot: put string map functionality in FairParGenericSet to avoid container classes
- fix current tasks etc to be fairroot conform

End

Thanks for Your Attention!

ALL.PAR

```
##################################################################
# Digitization parameters for MVD
# Format:
# parameter_name:parameter_type(i-integer, d-double) parameter_value
# PndMvdDigiPar
# Description of parameters:
##################################################################
##################################################################
# Digitization parameters for RECTANGULAR Mvd strip sensors
##################################################################
# threshold set to 5*sigmaNoise (noise rate ~ DPM signals rate)
[MVDStripDigiParRect]
top_pitch:Double_t 0.0130
bot_pitch:Double_t 0.0130
orient:Double_t 1.570796327
skew:Double_t -1.570796327
top_anchor_x:Double_t -3.3345
top_anchor_y:Double_t 1.6705
bot_anchor_x:Double_t -3.3345
bot_anchor_y:Double_t 1.6705
nr_fe_channels:Int_t 128
nr_fe_top:Int_t 4
nr_fe_bottom:Int_t 2
charge_threshold:Double_t 5000
charge_noise:Double_t 1000
QCloudSigma:Double_t 0.000581
sens_Type:Text_t Rect
fe_Type:Text_t APV25
fe_BusClock:Double_t 30
cluster_mod:Int_t 0
cluster_mean:Int_t 0
cluster_radchan:Int_t 2
cluster_radtime:Int_t 0
cluster_corrchargecut:Double_t 12000.
cluster_singlechargecut:Double_t 3000.
chargeconv_method:Int_t 1
##################################################################
```

PndSdsPixelDigiPar.h

```
class PndSdsPixelDigiPar : public FairParGenericSet
  {
    public :
    PndSdsPixelDigiPar (const char* name="PndSdsParTest",
                        const char* title="PndSds pixel digi parameter",
                        const char* context="TestDefaultContext");
    ~PndSdsPixelDigiPar(void){};
    void clear(void){};
    void putParams(FairParamList* list);
    Bool_t getParams(FairParamList* list);

    void Print();

    Double_t GetXPitch()        const {return fDimX;}
    Double_t GetYPitch()        const {return fDimY;}
    Double_t GetThreshold()     const {return fThreshold;}
    Double_t GetNoise()         const {return fNoise;}
    Double_t GetQCloudSigma()   const {return fCSigma;}
    Int_t GetFECols()           const {return fFeCols;}
    Int_t GetFERows()           const {return fFeRows;}
    Double_t GetClustRadius()   const {return fRadius;}
    Double_t GetFeBusClock()    const {return fFeBusClock;}
    Double_t GetTimeStep()    const {return (1./fFeBusClock * 1000.);} ///< Time step of one clock cycle in ns
    Int_t GetChargeConvMethod() const  {return fChargeConvMethod;}
    Double_t GetPixelSorterCellWidth() const {return fPixelSorterCellWidth;}
    Int_t GetPixelSorterNumberOfCells() const {return fPixelSorterNumberOfCells;}

    void SetXPitch(Double_t x)        {fDimX = x;}
    void SetYPitch(Double_t x)        {fDimY = x;}
    void SetThreshold(Double_t x)     {fThreshold = x;}
    void SetNoise(Double_t x)         {fNoise = x;}
    void SetQCloudSigma(Double_t x)   {fCSigma = x;}
    void SetFECols(Int_t x)           {fFeCols = x;}
    void SetFERows(Int_t x)           {fFeRows = x;}
    void SetClustRadius(Double_t x)   {fRadius=x;}
    void SetFeBusClock(Double_t x)      {fFeBusClock = x;}
    void SetChargeConvMethod(Int_t x) {fChargeConvMethod = x;}

    void SetPixelSorterCellWidth(Double_t x) {fPixelSorterCellWidth = x;}
    void SetPixelSorterNumberOfCells(Int_t x) {fPixelSorterNumberOfCells=x;}

  private:
    // Pixel Parameters
    Int_t fFeCols;            // Colums read per Frontend
    Int_t fFeRows;            // Rows read per Frontend
    Double_t fDimX;           // Pixel cell size X
    Double_t fDimY;           // Pixel cell size Y
    Double_t fRadius;         // Clusterfinder search radius (channel numbers)
    Double_t fThreshold;      // Discriminator Threshold (electrons)
    Double_t fNoise;          // Gaussian electronics noise including threshold dispersion (electrons)
    Double_t fCSigma;         // Gaussian charge cloud smearing
    Double_t fFeBusClock;   // Frontend bus clock to determin noise rate
    Int_t fChargeConvMethod;  // 0: ideal conversion; 1: TOT calculation
    Double_t fPixelSorterCellWidth; // Parameter for TimeStamp Sorter
    Int_t fPixelSorterNumberOfCells; // Parameter for TimeStamp Sorter
    //Text_t fSensName;         // Sensor name
    //Text_t fFeName;           // Frontend name
```

PndSdsHybridHitProducer.h

```
Double_t flx; //pixel width in x;
Double_t fly; //pixel width in y;
Double_t fthreshold; //pixel threshold in electrons
Double_t fnoise; //pixel noise in electrons
Double_t fqsigma; //gaussian charge cloud spread
Int_t    fcols; //pixel columns in one FE
Int_t    frows; //pixel rows in one FE
Int_t fPixelHits;
/  FairMCEventHeader* fMcEventHeader;
```

PndSdsHybridHitProducer.cxx

```
if(fOverwriteParams==kTRUE){
  fDigiPar->SetXPitch(flx);
  fDigiPar->SetYPitch(fly);
  fDigiPar->SetThreshold(fthreshold);
  fDigiPar->SetNoise(fnoise);
  fDigiPar->SetFECols(fcols);
  fDigiPar->SetFERows(frows);
  fDigiPar->setInputVersion(ana->GetRunId(),1);
  fDigiPar->setChanged();
  if(fVerbose>0) Info("Init","RTDB updated");
}

if(fVerbose>2) fDigiPar->Print();

flx = fDigiPar->GetXPitch();
fly = fDigiPar->GetYPitch();
fthreshold = fDigiPar->GetThreshold();
fnoise = fDigiPar->GetNoise();
fcols = fDigiPar->GetFECols();
frows = fDigiPar->GetFERows();
fqsigma = fDigiPar->GetQCloudSigma();
```

Concept Classes

```
#ifndef PNDCONFIGPARAMETERS_H_
#define PNDCONFIGPARAMETERS_H_

#include <map>
#include <string>

#ifndef __CINT__
#include "boost/property_tree/ptree.hpp"
#endif /* __CINT __ */

class PndConfigParameters {
  // usually we would just have the ptree here
  // unfortunately root cint is not capable to parse the ptree header
  // so this simpler option map was "developed"...
  std::map<std::string, std::string> config_parameters;

public:
  PndConfigParameters();
  virtual ~PndConfigParameters();

#ifndef __CINT__
  void insertParameters(const std::string &name_base,
      const boost::property_tree::ptree &config_parameters_);
  void setParameters(
      const boost::property_tree::ptree &config_parameters_);

  boost::property_tree::ptree getConfigParametersPropertyTree() const;
#endif /* __CINT __ */

  template<typename T> T getParameterValueByName(
      const std::string &param_name) const;
};

#endif /* PNDCONFIGPARAMETERS_H_ */
```

Concept Classes

```
#include "PndConfigParametersIO.h"
#include <fstream>

#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/xml_parser.hpp>

using boost::property_tree::ptree;

PndConfigParameters PndConfigParametersIO::readConfigFromFile(
    const std::string &config_url) const {
  // Create an empty property tree object
  boost::property_tree::ptree pt;

  // read the config file
  read_xml(config_url, pt);

  // create the simulation and reconstruction parameter object
  PndConfigParameters sim_reco_params;
  sim_reco_params.setParameters(pt);

  return sim_reco_params;
}

void PndConfigParametersIO::writeConfigToFile(
    const PndConfigParameters &sim_reco_params,
    const std::string &output_file_url) const {
  std::fstream file(output_file_url.c_str(), std::ios_base::out);
  write_xml(file, sim_reco_params.getConfigParametersPropertyTree());
  file.close();
}
```