Projet PSPA
Introduction d'un nouveau logiciel

Guy Le Meur

juin 2013

Je donne ci-après la procédure actuelle pour introduire un nouveau logiciel dans PSPA, que je désigne dans ce qui suit par « nouveauLogiciel »

Le répertoire WorkingArea doit contenir un lien symbolique vers l'exécutable de ce logiciel (soit, par exemple nouveauLogiciel.exe). Ce logiciel est censé lire son (ou ses) fichier(s) de données dans le même répertoire et il produira ses fichiers de sortie en cet endroit.

Les interventions suivantes sont à effectuer dans les répertoires 'sources/controler/include' et 'sources/controler/src'

1 Compléter la classe nomDeLogiciel

Dans le fichier nomDeLogiciel.h ajouter nouveauLogiciel à l'énumération 'Logiciel ' et dans le fichier nomDeLogiciel.cc mettre à jour les méthodes fromString et toString et getWtColor, c'est-à-dire :

dans fromString ajouter :

```
else if ( s == "nouveauLogiciel" ) return nouveau;Logiciel
```

dans toString ajouter :

```
case nouveauLogiciel : { return "nouveauLogiciel"; }
```

dans getWtColor ajouter :

```
case nouveauLogiciel : { return "[un code de couleur]"; }
```

Cela a pour seul effet d'ajouter le nouveau logiciel dans la liste du comboBox des logiciels dans le panneau sections' de l'interface. Il faut maintenant programmer l'interface.

2 programmer la classe softwareNouveauLogiciel

Il s'agit de créer une classe softwareNouveauLogiciel, héritant de la classe abstractSoftware. Il faut fournir les deux fichiers softwareNouveauLogiciel.h et softwareNouveauLogiciel. Le mieux est de procéder en s'aidant des interfaces déjà réalisées.

Les canevas en sont les suivants :

softwareNouveauLogiciel.h :

```
#ifndef  SOFTWARENOUVEAULOGICIEL_SEEN
#define  SOFTWARENOUVEAULOGICIEL_SEEN
//  Created by  ........................
//  Copyright .......

#include "abstractSoftware.h"

class nouveauLogiciel : public abstractSoftware
{

protected :

  / ..........

public :

  nouveauLogiciel();
  nouveauLogiciel(string inputFileName,
globalParameters* globals, dataManager* );
  virtual ~softwareTest() {;}
  virtual bool createInputFile( particleBeam* beamBefore,
            unsigned int numeroDeb,
                  unsigned int numeroFin,  string workingDir);
  virtual bool  execute(string workingDir);
  virtual bool buildBeamAfterElements(string workingDir);
};
#endif
```

softwareNouveauLogiciel.cc :

constructeurs :

```
nouveauLogiciel::nouveauLogiciel() : abstractSoftware()
{
  nameOfSoftware_ = nomDeLogiciel("nouveauLogiciel");
}

nouveauLogiciel::nouveauLogiciel(string inputFileName,
                 globalParameters* globals, dataManager* dt) :
                  abstractSoftware( inputFileName, globals, dt)
{
  nameOfSoftware_ = nomDeLogiciel("nouveauLogiciel");
// enregistrement des élements acceptés par
// le logiciel, exemple :
 registerElement(nomdElements::drift,TBoolOk);
  ..................................

// enregistrement des élements compatibles mais ignorés
// par le logiciel, exemple :
  registerElement(nomdElements::fit,TBoolIgnore);

  ..................................

}
```

méthode : createInputFile

```
bool nouveauLogiciel::createInputFile(
        particleBeam* beamBefore,
        unsigned int numeroDeb, unsigned int numeroFin,
        string workingDir)
{
  // recuperer :
  // * les paramètres globaux nécessaires au nouveauLogiciel
  //   (attribut de abstractSoftware : globParamPtr_ (classe
  //   globalParameters)
  //   exemple : double wt =
   //                      globParamPtr_->getIntegrationStep()

  // * les caractérisitiques du faisceau de début de section
  //   (beamBefore) a l'aide des méthodes
// de la classe particleBeam,
  //   exemple : int nb = beamBefore->getNbParticles()

  // * les données relatives aux éléments de la section au
  //   format spécifiqueà nouveauLogiciel,
// dans un flux ofstream :

  // fabriquer le fichier d'input :
  string name = workingDir + inputFileName_;
  ofstream outfile;
  outfile.open(name.c_str(), ios::out);
  if (!outfile) {
    dataManager_->consoleMessage(" nouveauLogiciel::
        createInputFile : error opening output stream " );
    return false;
  }
  // mettre dans outfile d'éventuelles
  // informations spécifiques
  for ( k = numeroDeb ; k <= numeroFin ; k++)
    {
      abstractElement* elPtr = dataManager_->
                        getElementPointerFromNumero(k);
      if (elPtr) {
outfile << elPtr->nouveauLogicielOutputFlow();
      }
      // mettre dans outfile d'éventuelles
     // informations spécifiques
    }
  // mettre dans outfile d'éventuelles
  // informations spécifiques
  return true;
}
```

méthode : execute

```
bool  nouveauLogiciel::execute(string workingDir) {
  bool ExecuteStatus = true;

  ostringstream sortie;

  string nouveauLogicielJob = workingDir +
                                    "nouveauLogiciel.exe";
  // parmelaJob += .....

  string resultOfRun;
  bool success = launchJob(nouveauLogicielJob,resultOfRun);

  if ( !success ) {
    sortie << " launching of nouveauLogiciel failed " << endl;
    ExecuteStatus = false;
  } else {
    sortie << " successful launching
                        of nouveauLogiciel " << endl;

    // ..........................

  dataManager_->consoleMessage(sortie.str());
  return ExecuteStatus;
}
```

méthode : buildBeamAfterElements

```
bool  nouveauLogiciel::buildBeamAfterElements(string
                    workingDir)
{
  bool result = true;

  // vérifications éventuelles


  // on initialise une nouvelle sortie diagnostic faisceau
  particleBeam* newDiag = dataManager_->
                              updateCurrentDiagnostic(true);
  // lecture des sorties nouveauLogiciel
  // verifications

  // newDiag->setXXX(....)
}
```