```
/*************************************************************************
 *  Logiciel portageMM2CPP de portage de code ModelMaker en langage C++      *
 *  Copyright INRA, février 2006                                           *
 *************************************************************************/

/*************************************************************************
 * Licence :

Le logiciel portageMM2CPP est une réécriture en langage C++ du modèle
écrit dans le fichier ModelMaker "modele2.mod" (situé dans le répertoire
"leSourceModelMaker"). Voir informations dans ../laDocumentation/help.

Ce logiciel est régi par la licence CeCILL soumise au droit français et
respectant les principes de diffusion des logiciels libres. Vous pouvez
utiliser, modifier et/ou redistribuer ce programme sous les conditions
de la licence CeCILL telle que diffusée par le CEA, le CNRS et l'INRIA
sur le site "http://www.cecill.info".

En contrepartie de l'accessibilité au code source et des droits de copie,
de modification et de redistribution accordés par cette licence, il n'est
offert aux utilisateurs qu'une garantie limitée.  Pour les mêmes raisons,
seule une responsabilité restreinte pèse sur l'auteur du programme,  le
titulaire des droits patrimoniaux et les concédants successifs.

A cet égard  l'attention de l'utilisateur est attirée sur les risques
associés au chargement,  à l'utilisation,  à la modification et/ou au
développement et à la reproduction du logiciel par l'utilisateur étant
donné sa spécificité de logiciel libre, qui peut le rendre complexe à
manipuler et qui le réserve donc à des développeurs et des professionnels
avertis possédant  des  connaissances  informatiques approfondies.  Les
utilisateurs sont donc invités à charger  et  tester  l'adéquation  du
logiciel à leurs besoins dans des conditions permettant d'assurer la
sécurité de leurs systèmes et ou de leurs données et, plus généralement,
à l'utiliser et l'exploiter dans les mêmes conditions de sécurité.

Le fait que vous puissiez accéder à cet en-tête signifie que vous avez
pris connaissance de la licence CeCILL, et que vous en avez accepté les
termes.

Les fichiers Licence_CeCILL_V2-fr.txt et Licence_CeCILL_V2-en.txt
du répertoire ../LICENCE fournissent le texte de la licence,
dans sa version 2, en langue française et en langue anglaise.
(ces 2 fichiers sont aussi directement dans le répertoire courant)

 *************************************************************************/

/*************************************************************************
```

```
 *
 * Fichier     : genericitePortageMM2CPP.cpp
 *
 * Auteur(s)   : Nathalie Rousse, Nathalie.Rousse@toulouse.inra.fr
 *               de l'INRA - Institut National de la Recherche Agronomique -
 *               (département MIA, UMR AGIR, http://www.modelia.org).
 *
 * Description :
 *
 * Ce fichier fait partie du code source C++ du logiciel portageMM2CPP.
 *
 * Il en contient la partie générique qui "se veut" indépendante du modèle
 * traité (corps des classes). Voir ../laDocumentation/help, en particulier
 * "Présentation générale" et "Spécification".
 *
 ************************************************************************
 * Historique :
 *
 * 03/02/06, Nathalie Rousse : création du fichier.
 * 24/10/07, Nathalie Rousse : modification mineure (Fonction simuler : modification contenu de la boucle d'execution d
e la simulation).
 * 25/10/07, Nathalie Rousse : modification mineure m_bool (utilisation du type predefini "bool" au lieu d'utiliser "in
t").
 *
 * 02/01/08, Nathalie Rousse : modification m_linearInterpolation (ajout prise en compte des entités ModelMaker qui son
t calculées par interpolation linéaire à partir de données lues dans un fichier de données).
 * 02/01/08, Nathalie Rousse : modification m_integrale.
 *
 ************************************************************************/


/* modeDemploi
 * Instruction s'adressant à quelqu'un qui est en train de construire son propre logiciel C++ relativement à son propre
 modèle ModelMaker : dans tout le code de ce fichier, consulter et suivre les consignes/indications marquées du label "
modeDemploi". Voir aussi "Mode d'emploi" dans le fichier "help".
 */

#include <iostream>
#include <fstream>
using namespace std;

#include "genericitePortageMM2CPP.h"


/************************************************************************
 *
 * corps de la classe entityInstantanee
```

```
 *
 ******************************************************************************/

entityInstantanee::entityInstantanee(void){ initialiser((typeValeur)0.0); } // par défaut valeur nulle
entityInstantanee::entityInstantanee(typeValeur iv){ initialiser(iv); }

void entityInstantanee::initialiser(typeValeur iv ){
        value=iv; initialValue=iv;
}

typeValeur entityInstantanee::getValue(void){ return value; }
typeValeur entityInstantanee::getInitialValue(void){ return initialValue; }


/******************************************************************************
 *
 * corps de la classe entityAvecDerivee
 *
 ******************************************************************************/

entityAvecDerivee::entityAvecDerivee( void ){
        initialiser((typeValeur)0.0, (typeValeur)0.0); // par défaut valeur et dérivée nulles
}
entityAvecDerivee::entityAvecDerivee( typeValeur iv ){
        initialiser(iv, (typeValeur)0.0); // par défaut dérivée nulle
}
entityAvecDerivee::entityAvecDerivee( typeValeur iv, typeValeur iddt ){
        initialiser(iv, iddt );
}

void entityAvecDerivee::initialiser (typeValeur iv){
        initialiser(iv, (typeValeur)0.0); // dérivée nulle par défaut
}
void entityAvecDerivee::initialiser(typeValeur iv, typeValeur iddt ){
        entityInstantanee::initialiser(iv); ddt=iddt;
}

/*
 * calcule/met à jour la valeur instantanée "value" en fonction de
 * sa d/dt ("ddt")
 */
void entityAvecDerivee::actualiser(typeTemps deltat){
        value = value + ddt * (typeValeur)deltat;
}


// m_linearInterpolation : nouvelle classe
```

```
/*****************************************************************************
 *
 * corps de la classe fichierDentree
 *
 *****************************************************************************/

/*
 * Constructeur et initialisations
 */

fichierDentree::fichierDentree( void ){
        initialiser();
        setNomFichier( "nomFichierNaPasEteDonne" ); // "pas" de nom de fichier par défaut
}

/* Initialisation par défaut */
void fichierDentree::initialiser( void ){
        estValide = false; // par défaut
        estPreparePourLecture = false; // par défaut
}

/* Initialise nomFichier, estValide, estPreparePourLecture */
void fichierDentree::initialiser( char *nom ){

        initialiser();
        setNomFichier( nom ); // initialisation du nom du fichier

        /*
         * Définit si le fichier est ou non valide (existe, peut être ouvert...) :
         * pour cela l'ouvre et le referme.
         */

        /* avant d'appeler preparerLecture, estValide est artificiellement
         * mis à true pour forcer l'exécution du traitement de préparation */
        estValide = true;
        preparerLecture(); // traitement de préparation

        /* Détermination de estValide */
        if ( estPreparePourLecture == true ){
                estValide = true;

                /* vérifie de plus qu'on peut lire au moins une donnée dans le fichier */
                float donneeLue;
                if ( lireFloat( &donneeLue ) == false ){ // donneeLue pas bien lue
                        estValide = false;
                }
        } else {
```

```
                estValide = false;
        }

        terminerLecture(); // notamment remet estPreparePourLecture à false
}

/*
 * preparerLecture ouvre le fichier nomFichier et en rend compte dans estPreparePourLecture.
 * Il faut appeler preparerLecture avant de faire (succession d')appel(s) de lireFloat().
 */
void fichierDentree::preparerLecture(){

        if ( estValide == false ){ // on ne fait rien sur le fichier
                estPreparePourLecture = false;
        } else { // estValide est true

                // tentative d'ouverture du fichier
                inputFile.open( nomFichier );

                if ( !inputFile ){ // le fichier n'a pas été ouvert correctement
                        estPreparePourLecture = false;
                } else { // le fichier a été ouvert correctement
                        estPreparePourLecture = true;
                }
        }
}

/*
 * terminerLecture ferme le fichier nomFichier (via inputFile) et en rend compte dans estPreparePourLecture.
 * Il faut appeler terminerLecture une fois fait(s) (succession d')appel(s) de lireFloat()
 */
void fichierDentree::terminerLecture(){
        inputFile.close();
        estPreparePourLecture = false;
}

bool fichierDentree::lireFloat( float *donneeLue ){
        float f;
        bool ret = false; // par défaut
        if ( estPreparePourLecture == true ){ // (fichier bien ouvert...)
                if ( inputFile >> f ){ // lecture de la donnée bien effectuée
                        *donneeLue = f;
                        ret = true;
                }
        }
        // si ret est false : la donnée n'a pas été lue ou n'a pas été correctement lue (fin de fichier...)
        return ret;
```

```
}

void fichierDentree::setNomFichier( char *nom ){
        strcpy( nomFichier, nom );
}
char * fichierDentree::getNomFichier(){
        return nomFichier;
}

void fichierDentree::setEstValide( bool estValideValue ){
        estValide = estValideValue;
}
bool fichierDentree::getEstValide(){
        return estValide;
}
void fichierDentree::setEstPreparePourLecture( bool estPreparePourLectureValue ){
        estPreparePourLecture = estPreparePourLectureValue;
}
bool fichierDentree::getEstPreparePourLecture(){
        return estPreparePourLecture;
}


// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * corps de la classe donneesAinterpoler
 *
 *****************************************************************************/

/*
 * Constructeur et initialisations
 */
void donneesAinterpoler::initialiser( char *nomFichier,
                                      typeIndice inbColonnes,
                                      typeIndice icolDonneeDeCtrl,
                                      typeIndice icolDonneeControlee ){
        nbColonnes = inbColonnes;
        colDonneeDeCtrl = icolDonneeDeCtrl;
        colDonneeControlee = icolDonneeControlee;

        fichierSource.initialiser( nomFichier ); // notamment instancie estValide

        /*
         * Vérification de la cohérence entre le nombre et les indices
         * de colonnes dans le fichier
         */
```

```
        bool coherenceNbEtIndicesColonnes = true; // par défaut
        if ( ( nbColonnes <=0 )
          || ( colDonneeDeCtrl <0 )
          || ( nbColonnes <= colDonneeDeCtrl )
          || ( colDonneeControlee <0 )
          || ( nbColonnes <= colDonneeControlee ) ){
                coherenceNbEtIndicesColonnes = false;
        }

        if ( coherenceNbEtIndicesColonnes == true ){

                /*
                 * Vérifie de plus qu'on peut lire au moins deux lignes dans le fichier
                 */
                float donneeDeCtrl; float donneeControlee; bool donneeBesoinNonTrouveeDansFichier;

                preparerLecture();

                /* Lit la 1ère ligne du fichier */
                lireDonneeDeCtrlEtDonneeControlee( &donneeDeCtrl, &donneeControlee, &donneeBesoinNonTrouveeDansFichier
);
                if ( donneeBesoinNonTrouveeDansFichier == false ){ // la ligne a bien été lue
                        /* Lit la 2nde ligne du fichier */
                        lireDonneeDeCtrlEtDonneeControlee( &donneeDeCtrl, &donneeControlee, &donneeBesoinNonTrouveeDans
Fichier );
                        if ( donneeBesoinNonTrouveeDansFichier == false ){ // la ligne a bien été lue
                                // vérification satisfaisante, ne rien faire de plus
                        } else { // 2nde ligne pas bien lue
                                setFichierSourceEstValide( false );
                        }
                } else { // 1ère ligne pas bien lue
                        setFichierSourceEstValide( false );
                }
                terminerLecture();

        } else { // coherenceNbEtIndicesColonnes est false

                setFichierSourceEstValide( false ); // pour signifier l'incohérence

#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                cout << endl << " ***** Fichier d'entités à interpoler ";
                cout << getNomFichierSource();
                cout << " : incohérence entre le nombre de colonnes et les indices de colonnes." << endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */
        }
}
```

```
char *donneesAinterpoler::getNomFichierSource( void ){
        return fichierSource.getNomFichier();
}
void donneesAinterpoler::setFichierSourceEstValide( bool estValideValue ){
        fichierSource.setEstValide( estValideValue );
}
bool donneesAinterpoler::getFichierSourceEstValide( void ){
        return ( fichierSource.getEstValide() );
}


void donneesAinterpoler::lireDonneeDeCtrlEtDonneeControlee( float *ptr_donneeDeCtrl,
                                                            float *ptr_donneeControlee,
                                                            bool *donneeBesoinNonTrouveeDansFichier ){
        *donneeBesoinNonTrouveeDansFichier = false; // par défaut
        float donneeLue;

        /* lecture des nbColonnes données de la ligne courante */
        for (int i = 0; i < nbColonnes; i++ ){

                /* lit une donnée dans le fichier */
                if ( fichierSource.lireFloat( &donneeLue ) == true ){
                // donneeLue bien lue
                        if ( i == colDonneeDeCtrl ){ // récupération de donneeDeCtrl
                                *ptr_donneeDeCtrl = donneeLue;
                        }
                        if ( i == colDonneeControlee ){ // récupération de donneeControlee
                                *ptr_donneeControlee = donneeLue;
                        }

                } else { // donneeLue mal lue
                // On considère que c'est parce que toutes les données ont été lues (fin de fichier)
                        *donneeBesoinNonTrouveeDansFichier = true;
                        fichierSource.terminerLecture();
                        i = nbColonnes; // pour forcer sortie de la boucle
                }
        }
}

void donneesAinterpoler::preparerLecture( void ){
        fichierSource.preparerLecture();
}
void donneesAinterpoler::terminerLecture( void ){
        fichierSource.terminerLecture();
}
```

```
// m_linearInterpolation : nouvelle classe
/*****************************************************************************
 *
 * corps de la classe entityInterpolee
 *
 *****************************************************************************/

/*
 *
 * Constructeur et initialisations
 *
 */

entityInterpolee::entityInterpolee( void ){
        // par défaut "pas" de nom de fichier
        initialiser( "nomFichierNaPasEteDonne", (typeIndice)2, (typeIndice)0, (typeIndice)1 );
}

entityInterpolee::entityInterpolee( char *nomFichier,
                                    typeIndice inbColonnes,
                                    typeIndice icolDonneeDeCtrl,
                                    typeIndice icolDonneeControlee ){
        initialiser( nomFichier, inbColonnes, icolDonneeDeCtrl, icolDonneeControlee );
}

void entityInterpolee::initialiser( char *nomFichier,
                                    typeIndice inbColonnes,
                                    typeIndice icolDonneeDeCtrl,
                                    typeIndice icolDonneeControlee ){

        /* Initialise le fichier (son nom, vérifie qu'on peut y lire deux lignes...) */
        lesDonneesAinterpoler.initialiser( nomFichier, inbColonnes, icolDonneeDeCtrl, icolDonneeControlee );

        /*
         * Initialisation de tPrec, tSuiv, valuePrec, valueSuiv
         * à partir de la 1ère lue dans le fichier
         * (quant à la 2ème ligne lue dans le fichier, voir commentaire plus bas)
         */
        float donneeDeCtrl; float donneeControlee;
        bool donneeBesoinNonTrouveeDansFichier;

        lesDonneesAinterpoler.preparerLecture();

        /* Lit la 1ère ligne du fichier */
        lesDonneesAinterpoler.lireDonneeDeCtrlEtDonneeControlee( &donneeDeCtrl, &donneeControlee, &donneeBesoinNonTrouv
eeDansFichier );
```

```
        if ( donneeBesoinNonTrouveeDansFichier == false ){
        // la ligne a bien été lue

                tPrec = (typeTemps)donneeDeCtrl; tSuiv = tPrec;
                valuePrec = (typeValeur)donneeControlee; valueSuiv = valuePrec;

                /* Lit la 2nde ligne du fichier */
                lesDonneesAinterpoler.lireDonneeDeCtrlEtDonneeControlee( &donneeDeCtrl,
                                &donneeControlee, &donneeBesoinNonTrouveeDansFichier );
                if ( donneeBesoinNonTrouveeDansFichier == false ){
                // la ligne a bien été lue

                        /* L'appel
                        decalerIntervalleInterpolationLineaire( (typeTemps)donneeDeCtrl, (typeValeur)donneeControlee );
                         * aurait pour effet d'affecter à tPrec la valeur lue dans la 1ère ligne
                         * et à tSuiv la valeur lue dans la 2nde ligne.
                         * Cet appel est mis en commentaire car l'affectation initiale nécessaire est :
                         * tSuiv = tPrec = valeur lue dans la 1ère ligne (et non pas 2nde ligne) et "value" corresponda
ntes
                         */

                } else { // 2nde ligne pas bien lue
                        desactiverFichierDonneesAinterpolerEstValide();
                        // ... bien que lesDonneesAinterpoler.initialiser s'était bien passé
                        // (où on avait vérifié qu'il y avait au moins 2 lignes dans le fichier)
                }

        } else { // 1ère ligne pas bien lue
                desactiverFichierDonneesAinterpolerEstValide();
                // ... bien que lesDonneesAinterpoler.initialiser (où on avait vérifié
                // qu'il y avait au moins 2 lignes dans le fichier) s'était bien passé ;
                // ou alors la dévalidation est une confirmation
        }
        lesDonneesAinterpoler.terminerLecture();

        if ( leFichierDonneesAinterpolerEstValide() == true ){
        // On poursuit l'initialisation

                /* Initialisation de tDernier
                 * à partir de la dernière ligne lue dans le fichier */
                donneeBesoinNonTrouveeDansFichier = false; // par défaut
                float tNew; float valueNew;
                lesDonneesAinterpoler.preparerLecture();
                while ( donneeBesoinNonTrouveeDansFichier == false ){
                        /* Lit une nouvelle ligne du fichier (une nouvelle donnée à interpoler) */
                        lesDonneesAinterpoler.lireDonneeDeCtrlEtDonneeControlee( &tNew, &valueNew, &donneeBesoinNonTrou
veeDansFichier );
```

```
                        if ( donneeBesoinNonTrouveeDansFichier == false ){
                                // la ligne a bien été lue => réitérer
                        } else { // ligne pas bien lue // *donneeBesoinNonTrouveeDansFichier est true
                                // On considère que c'est parce que toutes les données ont été lues (fin de fichier)
                                tDernier = tNew;
                                lesDonneesAinterpoler.terminerLecture();
                        }
                }
        }

        if ( leFichierDonneesAinterpolerEstValide() == true ){
        // On poursuit l'initialisation

                /* Vérification de l'ordre strictement croissant
                 * des instants lus dans le fichier */
                donneeBesoinNonTrouveeDansFichier = false; // par défaut
                float tNew; float valueNew;
                float tNewPrec;
                bool tNewPrecActif = false; bool arreter = false;
                lesDonneesAinterpoler.preparerLecture();
                while ( (donneeBesoinNonTrouveeDansFichier == false) && (arreter == false) ){
                        /* Lit une nouvelle ligne du fichier (une nouvelle donnée à interpoler) */
                        lesDonneesAinterpoler.lireDonneeDeCtrlEtDonneeControlee( &tNew, &valueNew, &donneeBesoinNonTrou
veeDansFichier );
                        if ( donneeBesoinNonTrouveeDansFichier == false ){
                                // la ligne a bien été lue

                                if ( tNewPrecActif == true ){ // vérification seulement quand au moins 2 lectures
                                        if ( tNew <= tNewPrec ){
                                        /* incohérence dans les instants lus dans le fichier
                                         * (liste non strictement croissante) */

                                                desactiverFichierDonneesAinterpolerEstValide();
#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                                                cout << endl << " ***** Fichier d'entités à interpoler ";
                                                cout << getNomFichierDonneesAinterpoler();
                                                cout << " : problème de liste d'instants dans le fichier non strictemen
t croissante." << endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */
                                                arreter = true;
                                        }
                                }
                                tNewPrec = tNew; tNewPrecActif = true; // pour suite

                        } else { // ligne pas bien lue // *donneeBesoinNonTrouveeDansFichier est true
                                lesDonneesAinterpoler.terminerLecture();
                                // c'est fini
```

```
                        }
                }
        }

        /* Préparation pour la suite */
        lesDonneesAinterpoler.preparerLecture(); // ne fait rien si estValide est false
}

/*
 *
 * Informations et vérifications et set
 *
 */

bool entityInterpolee::leFichierDonneesAinterpolerEstValide( void ){
        return lesDonneesAinterpoler.getFichierSourceEstValide();
}
void entityInterpolee::activerFichierDonneesAinterpolerEstValide( void ){
        lesDonneesAinterpoler.setFichierSourceEstValide( true );
}
void entityInterpolee::desactiverFichierDonneesAinterpolerEstValide( void ){
        lesDonneesAinterpoler.setFichierSourceEstValide( false );
}
char *entityInterpolee::getNomFichierDonneesAinterpoler( void ){
        return lesDonneesAinterpoler.getNomFichierSource();
}

typeTemps entityInterpolee::getPremierInstantDonneesAinterpoler( void ){
        return tPrec;
}

typeTemps entityInterpolee::getDernierInstantDonneesAinterpoler( void ){
        return tDernier;
}


int entityInterpolee::utilisabiliteDonneesAinterpoler( typeTemps tDebutDeLaSimulation, typeTemps tMaxDeLaSimulation ){

        int cr = CR_OK;

        if ( leFichierDonneesAinterpolerEstValide() == false ){
                // les contrôles d'initialisation (y compris nombre et indices colonnes) ont relevé des problèmes
                cr = CR_NOK_BLOQUANT;

        } else { // vérifications qui sont de l'ordre de l'avertissement (non bloquantes)

                /*
```

```
                 * Comparaison de tDebutDeLaSimulation et tMaxDeLaSimulation
                 * au 1er et au dernier instant des données du fichier (données à interpoler).
                 * Pour une interpolation nominale :
                 * tPrec <= tDebutDeLaSimulation < tMaxDeLaSimulation <= tDernier
                 * et sinon : on sera amené à extrapoler (à gauche ou à droite).
                 */

                if ( ( getPremierInstantDonneesAinterpoler() <= tDebutDeLaSimulation ) && ( tMaxDeLaSimulation <= getDe
rnierInstantDonneesAinterpoler() ) ){
                        // OK, cas nominal/attendu

                } else {
                        // on sera amené à extrapoler pour le fichier en question
                        cr = CR_NOK_AVERTISSEMENT;
                }
        }

#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
        // trace
        afficherProblemeUtilisabiliteDonneesAinterpoler( cr, tDebutDeLaSimulation, tMaxDeLaSimulation );
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */

        return cr;
}

void entityInterpolee::afficherProblemeUtilisabiliteDonneesAinterpoler( int crUtilisabiliteDonneesAinterpoler,
                                typeTemps tDebutDeLaSimulation, typeTemps tMaxDeLaSimulation ){

        if ( crUtilisabiliteDonneesAinterpoler == CR_NOK_BLOQUANT ){
                cout << endl;
                cout << " ***** Erreur pour le fichier d'entités à interpoler ";
                cout << getNomFichierDonneesAinterpoler();
                cout << ": les contrôles d'initialisation (y compris nombre et indices colonnes) ont relevé des problèm
es.";
                cout << endl;

        } else if ( crUtilisabiliteDonneesAinterpoler == CR_NOK_AVERTISSEMENT ){

                cout << endl;
                cout << " ***** Avertissement pour le fichier d'entités à interpoler ";
                cout << getNomFichierDonneesAinterpoler();
                cout << " -- comparaison des conditions de simulation [tDebutDeLaSimulation,tMaxDeLaSimulation]= ";
                cout << "[ " << tDebutDeLaSimulation << ", " << tMaxDeLaSimulation << " ] ";
                cout << "à l'intervalle de temps des données à interpoler du fichier ";
                cout << "[ " << getPremierInstantDonneesAinterpoler();
                cout << ", " << getDernierInstantDonneesAinterpoler() << " ] : ";
                cout << "on n'est pas dans le cas nominal, on sera amené à extrapoler (à gauche ou à droite).";
```

```
                cout << endl;
        }
}

/*
 *
 * Calculs et leur gestion
 *
 */

void entityInterpolee::decalerIntervalleInterpolationLineaire( typeTemps tNew, typeValeur valueNew ){
        tPrec = tSuiv; valuePrec = valueSuiv; // ...Suiv devient ...Prec
        tSuiv = tNew; valueSuiv = valueNew; // ...New devient ...Suiv
}

int entityInterpolee::calculerPenteInterpolationLineaire( typeValeur *pente ){
        int cr = CR_OK; // par défaut
        typeTemps ecartTemps = tSuiv - tPrec;

        if ( ecartTemps != 0.0 ){
                *pente = (valueSuiv - valuePrec) / (typeValeur)ecartTemps;
        } else {
                cr = CR_NOK_tentativeDivisionParZero;
        }
        return cr;
}

int entityInterpolee::interpolationLineaire( typeTemps tCourant, typeValeur *valueCourante ){

        /* Remarque : normalement tPrec et tSuiv encadrent tCourant au plus près :
         * tPrec <= tCourant < tSuiv (sinon, extrapolation linéaire) */

        typeValeur penteInterpolationLineaire;
        bool cr = calculerPenteInterpolationLineaire( &penteInterpolationLineaire );
        if ( cr == CR_OK ){
                *valueCourante = penteInterpolationLineaire * (typeValeur)( tCourant - tPrec ) + valuePrec;
        }
        return cr;
}

void entityInterpolee::actualiser( typeTemps tCourant,
                                   bool *donneeBesoinNonTrouveeDansFichier,
                                   int *crCalculInterpolationlineaire ){

        if ( tCourant < tPrec ){
                /* On ne fait rien, value sera issue d'extrapolation à gauche
                 * de (tPrec, valueprec) et (tSuiv, valueSuiv) */
```

```
        } else { // tPrec <= tCourant
                /* On avance dans le fichier des données à interpoler
                 * jusqu'à ce que tPrec et tSuiv encadrent tCourant au plus près */

                *donneeBesoinNonTrouveeDansFichier = false; // par défaut

                while ( ( tSuiv <= tCourant ) && ( *donneeBesoinNonTrouveeDansFichier == false ) ){

                        /* Lit une nouvelle ligne du fichier (une nouvelle donnée à interpoler) */

                        float tNew; float valueNew;
                        lesDonneesAinterpoler.lireDonneeDeCtrlEtDonneeControlee( &tNew, &valueNew, donneeBesoinNonTrouv
eeDansFichier );

                        if ( *donneeBesoinNonTrouveeDansFichier == false ){
                        // la ligne a bien été lue

                                /* prise en compte de la donnée à interpoler lue */
                                decalerIntervalleInterpolationLineaire( (typeTemps)tNew, (typeValeur)valueNew );

                        } else { // ligne pas bien lue // *donneeBesoinNonTrouveeDansFichier est true

                                // On considère que c'est parce que toutes les données ont été lues (fin de fichier)
                                lesDonneesAinterpoler.terminerLecture();

                                /* Il ne faut plus avancer, ce qui sera le cas puisque
                                 * *donneeBesoinNonTrouveeDansFichier est true => fin du while.
                                 * value sera issue d'extrapolation à droite
                                 * de (tPrec, valuePrec) et (tSuiv, valueSuiv) */
                        }

                } // tPrec <= tCourant < tSuiv
        }

        /*
         * Calcul par interpolation de (tPrec, valuePrec) et (tSuiv, valueSuiv)
         * (qui dans certains cas peut être une extrapolation à gauche ou à droite)
         */
        *crCalculInterpolationlineaire = interpolationLineaire( tCourant, &value );

        /* Cette procédure peut conclure à une anomalie bloquante (cf indicateur crCalculInterpolationlineaire) qui ser
a prise en compte en provoquant la fin du logiciel. */
};


/*****************************************************************************
```

```
 *
 * le corps de la classe horlogeLocale
 *
 ***************************************************************************/

horlogeLocale::horlogeLocale(void ){
        initialiser( (typeTemps)INSTANT_INITIAL_DEFAUT, (typeTemps)DELTA_T_DEFAUT );
}
horlogeLocale::horlogeLocale( typeTemps it){
        initialiser( it, (typeTemps)DELTA_T_DEFAUT );
}

horlogeLocale::horlogeLocale( typeTemps it, typeTemps deltatValue){
        initialiser( it, deltatValue);
}

void horlogeLocale::initialiser( void ){ // initialisation par défaut
        initialiser( (typeTemps)INSTANT_INITIAL_DEFAUT, (typeTemps)DELTA_T_DEFAUT );
}

void horlogeLocale::initialiser( typeTemps it){
        initialiser( it, (typeTemps)DELTA_T_DEFAUT );
}

void horlogeLocale::initialiser( typeTemps it, typeTemps deltatValue){
        tInitial = it; deltat = deltatValue;
        retourAinstantInitial(); // ramène l'instant courant "t" à tInitial
}

/* redonne à l'instant courant la valeur tInitial */
void horlogeLocale::retourAinstantInitial(void){
        // m_integrale : erreur détectée lors des vérifications des calculs -> correction de l'ancienne ligne :
        // t = tInitial - deltat; // "-deltat" en vue de la 1ère incrémentation
        // remplacée par :
        t = tInitial;
}

typeTemps horlogeLocale::getInstantCourant(void){ return t; }
typeTemps horlogeLocale::getInstantInitial(void){ return tInitial; }
typeTemps horlogeLocale::getPasDeTemps(void){ return deltat; }

/* change la valeur de tInitial (sans toucher l'instant courant) */
void horlogeLocale::setInstantInitial(typeTemps it){
        tInitial = it;
}

void horlogeLocale::incrementerHorloge(void){
```

```
        t = t + deltat;
}

void horlogeLocale::afficherConfigurationHorloge(void){
        cout << " ***** les données horloge locale : " << endl;
        cout << "l'instant initial est : tInitial=" << tInitial << " s, " << endl;
        cout << "le pas de temps (plus petit incrément) est : deltat=" << deltat << " s." << endl;
        cout << "(valeur 'actuelle' de l'instant courant : t=" << t << " s.)" << endl << endl;
}


/*****************************************************************************
 *
 * corps de la classe genericSystem
 *
 *****************************************************************************/

/*****************************************************************************
 * Constructeurs, initialisations
 *****************************************************************************/

// Initialisation par défaut
void genericSystem::initialiser(void){

        /* horloge locale */
        time.initialiser(); // initialisation aux valeurs par défaut

        /* simulation */
        initialiserDonneesSimulation();
}

// Initialisation avec configuration de l'horloge locale
void genericSystem::initialiser( typeTemps it, typeTemps deltatValue){

        /* horloge locale */
        time.initialiser( it, deltatValue );

        /* simulation */
        initialiserDonneesSimulation();
}
// Initialisation par défaut des données de simulation
void genericSystem::initialiserDonneesSimulation(void){

        tDebutDeLaSimulation = time.getInstantInitial();
        pasDeLaSimulation = time.getPasDeTemps() * (typeTemps)PAS_SIM_SUR_PAS_TEMPS_MIN;

        /* tMaxDeLaSimulation est mis ici à une valeur cohérente qui n'est
```

```
         * pas définitive (sera changée) : il est calculé à chaque
         * démarrage de simulation */
        tMaxDeLaSimulation = tDebutDeLaSimulation + (typeTemps)3.0 * pasDeLaSimulation;
}


/*****************************************************************************
 * Les méthodes de simulation
 *****************************************************************************/

/* modeDemploiEvolutions
 * Il peut être besoin d'implémenter de nouvelles fonctions de simulation : pour sa propre utilisation, quelqu'un peut
être amené à écrire une nouvelle méthode "simuler" publique (méthode "chapeau" appelant la méthode de simulation de bas
e "simuler" privée).
 */

/*
 * La methode de simulation "de base".
 *
 * Remarque : toutes les autres methodes de simulation ("simuler" avec
 * divers parametres) appellent cette methode "simuler".
 *
 * Description du déroulement :
 *
 * La valeur de l'instant de début de simulation
 * (= valeur de l'instant courant de l'horloge en entree de la fonction)
 * est enregistree dans "tDebutDeLaSimulation".
 * La valeur de l'instant de fin de simulation
 * (= "tDebutDeLaSimulation" + "dureeDeLaSimulation")
 * est enregistree dans "tMaxDeLaSimulation".
 *
 * Boucle d'appel du modele,
 * ie { time.incrementerHorloge(); actualiserIncrement(); } :
 * - pour t (instant courant de l'horloge) allant de "tDebutDeLaSimulation"
 *   à "tMaxDeLaSimulation",
 * - avec, a chaque pas de la boucle, increment de t de la valeur deltat (de
 *   l'horloge).
 *
 * Les resultats de la simulation (ie l'etat du modele) sont affiches avec
 * comme periode "pasDeLaSimulation", entre les instants de debut et de fin
 * de simulation ("tDebutDeLaSimulation" et "tMaxDeLaSimulation").
 *
 */
int genericSystem::simuler(void){

        int cr = CR_OK; // par défaut
```

```
        /* enregistrement/calcul
         * de tDebutDeLaSimulation et tMaxDeLaSimulation */
        tDebutDeLaSimulation = time.getInstantInitial();
        tMaxDeLaSimulation = tDebutDeLaSimulation + dureeDeLaSimulation;

        /*
         * vérifications
         */

        if ( tDebutDeLaSimulation > tMaxDeLaSimulation ){
#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                cout << endl << " ***** [genericSystem::simuler] : CR_NOK_BornesIncoherentes" << endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */
                return CR_NOK_BornesIncoherentes;
        }

        if ( pasDeLaSimulation <= 0.0 ){
#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                cout << endl << " ***** [genericSystem::simuler] : CR_NOK_PasSimulNegatif" << endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */
                return CR_NOK_PasSimulNegatif;
        }

        /*
         * Dans la durée "pasDeLaSimulation", il faut que le "pas de temps de
         * base" (deltat de l'horloge) contienne au moins
         * PAS_SIM_SUR_PAS_TEMPS_MIN fois. Si ce n'est pas le cas, alors
         * pasDeLaSimulation est ici augmenté pour que ça le devienne.
         */
        // 26/10/07 m_bool
        bool pasDeLaSimulationRetouche = false; // indicateur pour information à l'écran
        typeTemps p = (typeTemps)PAS_SIM_SUR_PAS_TEMPS_MIN * time.getPasDeTemps();

        if ( pasDeLaSimulation < p ){
                pasDeLaSimulation = p;
                pasDeLaSimulationRetouche = true;
        }

        /*
         * Vérification de l'utilisabilité des fichiers des données interpolées :
         * vérification de leur autocohérence, vérification de la cohérence
         * entre leur contenu et les conditions de simulation...
         * Certaines vérifications sont bloquantes, d'autres donnent lieu à des avertissements.
         */
        int crUtilisabiliteDonneesAinterpoler = eil.utilisabiliteDonneesAinterpoler( tDebutDeLaSimulation, tMaxDeLaSimu
lation );
        if ( crUtilisabiliteDonneesAinterpoler == CR_NOK_BLOQUANT ){
```

```
#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                cout << endl << " ***** [genericSystem::simuler] : CR_NOK_InutilisabiliteFichiersDonneesAinterpoler" <<
 endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */
                return CR_NOK_InutilisabiliteFichiersDonneesAinterpoler;
        } /* else
         * Ou bien crUtilisabiliteDonneesAinterpoler==CR_OK : auquel cas tout va bien,
         * Ou bien crUtilisabiliteDonneesAinterpoler==CR_NOK_AVERTISSEMENT : auquel cas l'avertissement sera signalé à
l'utilisateur (affichage à l'écran plus bas).
         */

        // trace

        cout << endl;
        cout << " ***** DEBUT DE LA SIMULATION EFFECTUEE :" << endl << endl;
        cout << " ***** les conditions de la simulation effectuée : " << endl;
        cout << "la simulation sort des résultats pour des itérations ";
        cout << "écartées de pasDeLaSimulation=" << pasDeLaSimulation << " s, ";
        cout << "sur une durée allant de l'instant tDebutDeLaSimulation=" << tDebutDeLaSimulation << " s, ";
        cout << "jusqu'à l'instant tMaxDeLaSimulation=" << tMaxDeLaSimulation << " s." << endl;

        if ( pasDeLaSimulationRetouche == true ){
        cout << endl;
        cout << " ***** Remarque : pour des questions de précision des calculs, ";
        cout << "pasDeLaSimulation a été modifié (augmenté) par rapport à la demande, ";
        cout << "de telle sorte que dans la durée pasDeLaSimulation, il contienne ";
        cout << "au moins PAS_SIM_SUR_PAS_TEMPS_MIN (=" << PAS_SIM_SUR_PAS_TEMPS_MIN << ") fois ";
        cout << "le pas de temps (plus petit incrément) deltat (=" << time.getPasDeTemps() << " s)." << endl;
        }

        // Affichage de la liste des noms des fichiers d'entités à interpoler
        cout << endl;
        if ( eil.nb <= 0 ){
                cout << " ***** Remarque : il n'est utilisé aucun fichier d'entités à interpoler.";
        } else {
                cout << " ***** Pour info, liste des fichiers d'entités à interpoler :" << endl;
                eil.afficherNomsFichiersDonneesAinterpoler();
        }
        cout << endl;

        if ( crUtilisabiliteDonneesAinterpoler == CR_NOK_AVERTISSEMENT ){
        cout << endl;
        cout << " ***** AVERTISSEMENT en conclusion de la comparaison des conditions de simulation ";
        cout << "(tDebutDeLaSimulation et tMaxDeLaSimulation) ";
        cout << "avec l'intervalle de temps de chacun des (éventuels) fichiers de données à interpoler : ";
        cout << "pour au moins un fichier de données à interpoler, ";
        cout << "on n'est pas dans le cas nominal et on sera amené à extrapoler (à gauche ou à droite).";
```

```
	cout << endl;
	}
	cout << endl;

	time.afficherConfigurationHorloge();

	afficherEntetesEntites();

	/*
	 * boucle d'exécution de la simulation
	 *
	 * Demarrage de la simulation a "t", l'instant courant de l'horloge.
	 * Arret de la simulation a "tMaxDeLaSimulation".
	 *
	 * Debut des affichages des resultats de simulation a "tDebutDeLaSimulation".
	 * Fin des affichages des resultats de simulation a "tMaxDeLaSimulation".
	 */
	gSimu.initialiser( tDebutDeLaSimulation, pasDeLaSimulation );

	while ( ( time.getInstantCourant() <= tMaxDeLaSimulation ) && ( gSimu.finPrematuree == false ) ){

		/* Fait évoluer (incrémente) l'ensemble du système
		 * du pas de temps horloge "deltat" */
		time.incrementerHorloge();

		/* met à jour gSimu relativement au nouvel instant courant */
		gSimu.maj( time.getInstantCourant() );

		int crActualiserIncrement = actualiserIncrement();

		if ( crActualiserIncrement == CR_OK ){

			/* Sorties des resultats "tous les" pas de simulation pasDeLaSimulation */
			if ( gSimu.onEstSurUnPasDeLaSimulation == true ){
				afficherEntites();
			}

			// maj pour la suite
			gSimu.majPourlaSuite( time.getInstantCourant(), pasDeLaSimulation );

		} else {
			gSimu.finPrematuree = true;
			cr = crActualiserIncrement;
		}
	}

	if ( gSimu.finPrematuree == true ){
```

```
                cout << endl << " ***** FIN PREMATUREE DE LA SIMULATION à l'instant courant : ";
                cout << time.getInstantCourant();
                cout << " c'est à dire au jour : ";
                cout << ( time.getInstantCourant() / 86400.0 ) << endl;

        } else {
                cout << endl << " ***** FIN DE LA SIMULATION." << endl;
        }

        return cr;
}


/*
 * Une methode de simulation.
 *
 * Appelle la methode de simulation de base "simuler" de maniere que
 * la boucle d'exécution de la simulation se fasse ainsi :
 * Demarrage de la simulation a "t", l'instant courant de l'horloge (qui
 * n'est pas retouche ici avant de commencer la simulation).
 * Arret de la simulation apres une duree de "nbIterations"x"pasDeSimulation".
 * Periode d'affichage des resultats de la simulation (ie l'etat du modele) :
 * "pasSimulation", entre les instants de debut et de fin de la simulation.
 */
void genericSystem::simuler(int nbIterations, typeTemps pasSimulation){

        // trace
        cout << " ***** COMMANDE DE LA SIMULATION :" << endl << endl;
        cout << " ***** les conditions de simulation demandées : " << endl;
        cout << "il est demandé que la simulation sorte des résultats pour nbIterations="
                << nbIterations << " itérations successives, ";
        cout << "écartées de pasSimulation=" << pasSimulation << " s, ";
        cout << "à partir de l'instant courant." << endl << endl;

        pasDeLaSimulation = pasSimulation;
        dureeDeLaSimulation = (typeTemps)nbIterations * pasSimulation;
        /* tDebutDeLaSimulation et tMaxDeLaSimulation
         * seront definis dans "simuler" appele plus bas */

        int cr = simuler();
        if ( cr == CR_OK ){
                cout << endl;
                cout << "Fin de fonction simuler sans erreur." << endl << endl;
        } else {
                cout << endl;
                cout << "Fin de fonction simuler en erreur, code erreur : " << cr << "." << endl << endl;
        }
```

```
}


/*
 * Une methode de simulation.
 *
 * Modifie l'horloge : l'instant initial "tInitial" et l'instant courant
 * "t" de l'horloge sont mis à la valeur "tDebut".
 *
 * Puis appelle la methode de simulation de base "simuler" de maniere que
 * la boucle d'exécution de la simulation se fasse ainsi :
 * Demarrage de la simulation a "t", l'instant courant de l'horloge (qui
 * a prealablement ete ramene a "tDebut").
 * Arret de la simulation apres une duree de "dureeSimulation".
 * Periode d'affichage des resultats de la simulation (ie l'etat du modele) :
 * "pasSimulation", entre les instants de debut et de fin de la simulation.
 */
void genericSystem::simuler(typeTemps tDebut, typeTemps dureeSimulation, typeTemps pasSimulation ){

        cout << " ***** COMMANDE DE LA SIMULATION :" << endl << endl;
        cout << " ***** les conditions de simulation demandées : " << endl;
        cout << "il est demandé que la simulation sorte des résultats pour des itérations ";
        cout << "écartées de pasSimulation=" << pasSimulation << " s, ";
        cout << "sur une durée dureeSimulation=" << dureeSimulation << " s, ";
        cout << "à partir de l'instant tDebut=" << tDebut << " s ";
        cout << "(auquel l'horloge est recalee avant de commencer la simulation)." << endl << endl;

        // on recale l'horloge locale ("tInitial" et "t") à tDebut
        time.setInstantInitial( tDebut );
        time.retourAinstantInitial();

        pasDeLaSimulation = pasSimulation;
        dureeDeLaSimulation = dureeSimulation;
        /* tDebutDeLaSimulation et tMaxDeLaSimulation
         * seront definis dans "simuler" appele plus bas */

        int cr = simuler();
        if ( cr == CR_OK ){
                cout << "Fin de fonction simuler sans erreur." << endl;
        } else {
                cout << "Fin de fonction simuler en erreur, code erreur : " << cr << "." << endl;
        }
}


/*****************************************************************************
 *
```

```
 * le corps de la classe listeEntitesAvecDerivee
 *
 ****************************************************************************/

/* Construction/initialisation de la liste privée des entités avec dérivée */
listeEntitesAvecDerivee::listeEntitesAvecDerivee( void ){
        nb = 0;
        for ( int i=0; i<NB_MAX_EntitesAvecDerivee; i++){ liste[i] = NULL; }
        // 26/10/07 m_bool
        aucuneInitialisation = true;
}

/*
 * Entrer un élément (pointeur) dans "liste", la liste privée des entités
 * avec dérivée. L'élément n'est ajouté à la liste que s'il s'agit d'un
 * pointeur d'entité avec dérivée (si c'est un pointeur d'entité instantanée
 * ou d'entité interpolée, il n'est pas ajouté à la liste).
 */
int listeEntitesAvecDerivee::PL( pointeurDentityAvecDerivee ptr ){

        int cr = CR_OK; // par défaut

        if ( nb >= NB_MAX_EntitesAvecDerivee){
                // impossible d'ajouter l'entité à la liste qui est pleine
                cr = CR_NOK_DepassementDim;
        } else {
                liste[ nb ] = ptr; nb++;
                // 26/10/07 m_bool
                aucuneInitialisation = false;
        }
        return cr;
}
int listeEntitesAvecDerivee::PL( pointeurDentityInstantanee ptr ){

        /* une entité instantanée n'est pas saisie */

        // 26/10/07 m_bool
        aucuneInitialisation = false;
        return CR_OK;
};
int listeEntitesAvecDerivee::PL( pointeurDentityInterpolee ptr ){

        /* une entité interpolée n'est pas saisie */

        aucuneInitialisation = false;
        return CR_OK;
};
```

```
/*
 * Actualisation de la liste privée des entités avec dérivée :
 * Boucle de calcul des valeurs en fonction des dérivées, pour tous les
 * éléments de la liste "liste".
 * void en retour, car pas de gestion de cas où ça se passe mal (pas comme
 * prévu).
 */
void listeEntitesAvecDerivee::AL( typeTemps deltat ){

        entityAvecDerivee *ptr;

        // Mise à jour des valeurs "value" en fonction des dérivées "ddt"
        for ( int i=0; i<nb; i++){
                ptr = liste[ i ];
                ptr->actualiser( deltat );
        }
}


/*****************************************************************************
 *
 * le corps de la classe listeEntitesInterpolees
 *
 *****************************************************************************/

/* Construction/initialisation de la liste privée des entités interpolées */
listeEntitesInterpolees::listeEntitesInterpolees( void ){
        nb = 0;
        for ( int i=0; i<NB_MAX_EntitesInterpolees; i++){ liste[i] = NULL; }
        aucuneInitialisation = true;
}

/*
 * Entrer un élément (pointeur) dans "liste", la liste privée des entités
 * interpolées. L'élément n'est ajouté à la liste que s'il s'agit d'un
 * pointeur d'entité interpolée (si c'est un pointeur d'entité instantanée
 * ou d'entité avec dérivée, il n'est pas ajouté à la liste).
 * Retourne CR_NOK_FichierInvalide si le fichier de l'entité interpolée
 * a été identifié (lors de l'initialisation) comme non valide.
 */
int listeEntitesInterpolees::PL( pointeurDentityInterpolee ptr ){

        int cr = CR_OK; // par défaut

        if ( nb >= NB_MAX_EntitesInterpolees){
                // impossible d'ajouter l'entite à la liste qui est pleine
```

```
                cr = CR_NOK_DepassementDim;
        } else {

                /* l'entité interpolée est saisie quelle que soit son état
                 * de validité */
                liste[ nb ] = ptr; nb++;
                aucuneInitialisation = false;

                /* Si le fichier de l'entité interpolée est non valide,
                 * c'est signalé dans le compte rendu */
                if ( ptr->leFichierDonneesAinterpolerEstValide() == false ){
                        cr = CR_NOK_FichierInvalide;

                        // De plus affichage du nom du fichier (sous condition CLE_AVEC_TRACE_ECRAN_DONNEES) permettant
 d'identifier/repérer le fichier non valide
#ifdef CLE_AVEC_TRACE_ECRAN_DONNEES
                        cout << endl << " ***** Fichier d'entités à interpoler ";
                        cout << ptr->getNomFichierDonneesAinterpoler();
                        cout << " : non valide." << endl << endl;
#endif /* CLE_AVEC_TRACE_ECRAN_DONNEES */

                }
        }
        return cr;
}
int listeEntitesInterpolees::PL( pointeurDentityInstantanee ptr ){

        /* une entité instantanée n'est pas saisie */

        aucuneInitialisation = false;
        return CR_OK;
};
int listeEntitesInterpolees::PL( pointeurDentityAvecDerivee ptr ){

        /* une entité avec dérivée n'est pas saisie */

        aucuneInitialisation = false;
        return CR_OK;
};

/*
 * Actualisation de la liste privée des entités interpolées :
 * Boucle de calcul des valeurs interpolées, pour tous les éléments
 * de la liste "liste".
 * Retourne CR_OK si l'actualisation s'est bien passée pour tous
 * les éléments de la liste "liste" et sinon retourne un des
 * problèmes relevés (indique un seul problème même s'il y en a eu plusieurs).
```

```
 */
int listeEntitesInterpolees::AL( typeTemps tCourant ){

        int cr = CR_OK; // par défaut;

        pointeurDentityInterpolee ptr;

        bool donneeBesoinNonTrouveeDansFichier;
        int crCalculInterpolationlineaire;

        // Mise à jour des valeurs "value" par calcul d'interpolation
        for ( int i=0; i<nb; i++){
                ptr = liste[ i ];

                ptr->actualiser( tCourant, &donneeBesoinNonTrouveeDansFichier, &crCalculInterpolationlineaire );

                /* crCalculInterpolationlineaire rend compte d'une anomalie bloquante */
                if ( crCalculInterpolationlineaire != CR_OK ){
                        cr = crCalculInterpolationlineaire;
                }
                if ( donneeBesoinNonTrouveeDansFichier == true ){
                        // on ne fait rien

                        /* ATTENTION : alors qu'il a fallu lire une donnée dans le fichier, la donnée n'a pas été trouv
ée (a été mal lue), ce qui est interprété comme fin de fichier. Sachant qu'on avait au préalable vérifié dans donneesAi
nterpoler::initialiser qu'on pouvait lire au moins deux lignes dans le fichier, la fin de fichier n'est pas considérée
comme une erreur (on ne retourne pas CR_NOK_FIN_FICHIER) mais comme un cas où l'on sera amené à extrapoler à droite lor
s des calculs d'interpolation linéaire. TOUTEFOIS, il se peut que cela corresponde à un cas où le fichier n'est pas au
format attendu (nombre de colonnes, ligne incomplète...) */
                }
        }
        return cr;
}

/*
 * Vérification utilisabiliteDonneesAinterpoler pour la liste privée
 * des entités interpolées (boucle d'appel de utilisabiliteDonneesAinterpoler,
 * pour tous les éléments de la liste "liste").
 * Retourne CR_OK ou CR_NOK_AVERTISSEMENT ou CR_NOK_BLOQUANT.
 */
int listeEntitesInterpolees::utilisabiliteDonneesAinterpoler( typeTemps tDebutDeLaSimulation, typeTemps tMaxDeLaSimulat
ion ){

        int crPourTousLesFichiers = CR_OK; // par défaut
        bool auMoinsUnCrNokBloquant = false; // par défaut

        pointeurDentityInterpolee ptr;
```

```
        for ( int i=0; i<nb; i++){
                ptr = liste[ i ];

                int cr = ptr->utilisabiliteDonneesAinterpoler( tDebutDeLaSimulation, tMaxDeLaSimulation );
                if ( cr == CR_NOK_AVERTISSEMENT ){
                        crPourTousLesFichiers = CR_NOK_AVERTISSEMENT;
                } else if ( cr == CR_NOK_BLOQUANT ){
                        auMoinsUnCrNokBloquant = true;
                }
        }
        if ( auMoinsUnCrNokBloquant == true ){
                crPourTousLesFichiers = CR_NOK_BLOQUANT;
        }
        return crPourTousLesFichiers;
}

/*
 * Affichage de la liste des noms des fichiers de données à interpoler
 */
void listeEntitesInterpolees::afficherNomsFichiersDonneesAinterpoler(){

        pointeurDentityInterpolee ptr;

        for ( int i=0; i<nb; i++){
                ptr = liste[ i ];
                cout << ptr->getNomFichierDonneesAinterpoler() << endl;
        }
}


/*****************************************************************************
 *
 * le corps de la classe compteRenduInitialisation
 *
 *****************************************************************************/

compteRenduInitialisation::compteRenduInitialisation( void ){
        putValue( CR_PAS_DE_PB_RELEVE ); // par défaut
}

void compteRenduInitialisation::putValue( int value ){ crInitValue = value; }

int compteRenduInitialisation::getValue( void ){ return crInitValue; }

// 26/10/07 m_bool
bool compteRenduInitialisation::pasDePbInitReleve( void ){
```

```
        if ( getValue() == CR_PAS_DE_PB_RELEVE ){
                return true;
        } else {
                return false;
        }
}


/*****************************************************************************
 *
 * le corps de la classe gestionAvancementSimulation
 *
 *****************************************************************************/

/* pas de constructeur */

void gestionAvancementSimulation::initialiser( typeTemps tDebutDeLaSimulation,
                                               typeTemps pasDeLaSimulation ){

        /* C'est le tout 1er marqueur (à repérer), les "autres" suivront de
         * pasDeLaSimulation en pasDeLaSimulation */
        marqueurPasDeLaSimulation = tDebutDeLaSimulation;

        dernierInstantPasDeLaSimulation = tDebutDeLaSimulation - pasDeLaSimulation; // initialisation fictive

        finPrematuree = false;

        /* onEstSurUnPasDeLaSimulation est calculé dans maj */
}
void gestionAvancementSimulation::maj( typeTemps instantCourant ){

        /* Détermine si on est ou non sur un pas de simulation
         * pasDeLaSimulation (ie si on a atteint le marqueur) et
         * le prend en compte */
        if ( instantCourant >= marqueurPasDeLaSimulation ){
                // on est sur (vient juste de passer) un pas
                onEstSurUnPasDeLaSimulation = true;
        } else { // on n'est pas sur un pas
                onEstSurUnPasDeLaSimulation = false;
        }
}
void gestionAvancementSimulation::majPourlaSuite( typeTemps instantCourant, typeTemps pasDeLaSimulation ){

        if ( onEstSurUnPasDeLaSimulation == true ){
                dernierInstantPasDeLaSimulation = instantCourant; // maj
                marqueurPasDeLaSimulation = marqueurPasDeLaSimulation + pasDeLaSimulation; // maj pour la suite
        }
```

```
}

/*****************************************************************************
 *                                                                           *
 * Fin du fichier : genericitePortageMM2CPP.cpp                              *
 *                                                                           *
 *****************************************************************************/
```