

Modele Class

functions that a (sub-) model SHOULD re-implement

a constructor,
 -> calling the mother constructor Modele(nbvals, nbparams)
 -> filling names, extNames, paramNames
 and optionally paraLowBounds, paraUpBounds, and backSimulated
 virtual void derivatives(const vector<double> &x, vector<double> &dxdt, const double t);
 -> should calculate dxdt[] from the value of variables x[] and the current time t
 virtual void setBaseParameters();
 -> fill params[] with a default set of parameters
 virtual void initialise(long long _background = DEFAULT_BACKGROUND_VALUE);
 -> can change init[], params[], whatever, depending on an option (_background)

pre-implemented functions :

```
void simulate(double _sec_max, Evaluator* E = NULL);
-> uses the solver to simulate val[] from the current t to t + sec_max
-> the variables for which over(internalVarID) = true are replaced by data
-> each wished data-point (time, variable), pre-defined in the Evaluator E
is passed to E during simulation.
-> if saveKinetics = true, the tablecourse 'cinetique', is filled during simulation.
```

```
void applyOverride(vector<double> &x, double t);
-> fills the state vector x[vars] according to the overrides for time t
```

```
void clearOverride(vector<double> &x, vector<double> &dxdt);
-> fills x[]=0 and dxdt[]=0 for each overridden variable
to avoid that the solver gets confused in case x[] was not following dxdt[]
```

```
void initialiseDone();
deleteCinetique();
penalties = 0;
stopCriterionReached=false;
```

```
bool checkDivergence();
-> if any of x[] is NAN, INF, or > STOP_VAL, then stopCriterionReached = true;
penalties += 100 / (0.01 + log(t/100 + 1)) for each
```

```
string print();
```

Fields

```
int nbVars;
vector<double> init;
vector<string> names;
vector<int> extNames;
```

```
double dt;
double t;
vector<double> val;
```

```
double penalties;
bool stopCriterionReached;
```

```
int nbParams;
vector<string> paramNames;
vector<double> params;
```

```
bool parametersLoaded;
```

```
vector<double> paramUpBounds;
```

```
vector<double> paramLowBounds;
```

```
vector<long long> backSimulated;
override* currentOverride;
```

```
bool saveKinetics;
TableCourse* cinetique;

double print_all_secs;
```

Input/Output methods

```
Modele(int _nbVars, int _nbParams)
```

```
int getNbVars();
vector<double> getInit();
string getName(int internalID);
int getGlobalID(int internalID)
virtual int internValName(int idGlobalVariable);
virtual bool isVarKnown(int idGlobalVariable);

double getT();
virtual double getValue(int idGlobalVariable);
virtual void setValue(int idGlobalVariable, double val);
virtual void addValue(int idGlobalVariable, double val);
virtual void action(string name, double parameter);
virtual void action(string name, vector<double> parameters);
```

```
int getNbParams();
string getParamName(int i);
double getParam(int i);
vector<double> getParameters();
virtual void saveParameters(string file_name)
void setParam(int i, double v);
void setParameters(vector<double> &newParamSet);
virtual void setBaseParameters();
virtual void loadParameters(string file_name);
void setBaseParametersDone();
void loadParametersDone();
void needNewParameterSet();
double getLowerBound(int i);
void setBounds(int i, double vLow, double vHi);
void setBounds(vector<double> upVals, vector<double> lowVals);
double getUpperBound(int i);

virtual bool isSimuBack(long long idGlobalBack);
void setOverride(override* newOverride = NULL);
bool over(int indexLocal);
```

```
TableCourse getCinetique();
void deleteCinetique();
void newCinetiqueIfRequired();
void save_state(double _t);
void setPrintMode(bool z, double _print_all_secs);
```