AMICI

Generated by Doxygen 1.8.13

Contents

1 AMICI 0.1 General Documentation

1.1 Introduction

AMICI is a MATLAB interface for the SUNDIALS solvers CVODES (for ordinary differential equations) and IDAS (for algebraic differential equations). AMICI allows the user to specify differential equation models in terms of symbolic variables in MATLAB and automatically compiles such models as .mex simulation files. In contrast to the SUNDIALSTB interface, all necessary functions are transformed into native C code, which allows for a significantly faster numerical integration. Beyond forward integration, the compiled simulation file also allows for first and second order forward sensitivity analysis, steady state sensitivity analysis and adjoint sensitivity analysis for likelihood based output functions.

The interface was designed to provide routines for efficient gradient computation in parameter estimation of biochemical reaction models but is also applicable to a wider range of differential equation constrained optimization problems.

1.2 Availability

The sources for AMICI are accessible as

- Source tarball
- Source zipball
- GIT repository on github

Once you've obtained your copy check out the Installation

1.2.1 Obtaining AMICI via the GIT versioning system

In order to always stay up-to-date with the latest AMICI versions, simply pull it from our GIT repository and recompile it when a new release is available. For more information about GIT checkout their website

The GIT repository can currently be found at https://github.com/FFroehlich/AMICI and a direct clone is possible via

```
git clone https://github.com/FFroehlich/AMICI.git AMICI
```

1.2.2 License Conditions

1.3 Installation

If AMICI was downloaded as a zip, it needs to be unpacked in a convenient directory. If AMICI was obtained via cloning of the git repository, no further unpacking is necessary.

To use AMICI, start MATLAB and add the AMICI direcory to the MATLAB path. To add all toolbox directories to the MATLAB path, execute the matlab script

```
installToolbox.m
```

To store the installation for further MATLAB session, the path can be saved via

```
savepath
```

For the compilation of .mex files, MATLAB needs to be configured with a working C compiler. The C compiler needs to be installed and configured via:

```
mex -setup c
```

For a list of supported compilers we refer to the mathworks documentation: mathworks.de

The tools SUNDIALS and SuiteSparse shipped with AMICI do **not** require further installation.

AMICI uses the following packages from SUNDIALS:

CVODES: the sensitivity-enabled ODE solver in SUNDIALS. Radu Serban and Alan C. Hindmarsh. *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* American Society of Mechanical Engineers, 2005. PDF

IDAS

AMICI uses the following packages from SuiteSparse:

Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems. Timothy A. Davis, Ekanathan Palamadai Natarajan, *ACM Transactions on Mathematical Software*, Vol 37, Issue 6, 2010, pp 36:1 - 36:17. PDF

Algorithm 837: AMD, an approximate minimum degree ordering algorithm, Patrick R. Amestoy, Timothy A. Davis, Iain S. Duff, *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 381 - 388. PDF

Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm, Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 377 - 380. PDF

2 How to contribute

We are happy about contributions to AMICI in any form (new functionality, documentation, bug reports, ...).

Making code changes

When making code changes:

- Check if you agree to release your contribution under the conditions provided in `LICENSE`
- Start a new branch from `master`
- Implement your changes
- Submit a pull request
- Make sure your code is documented appropriately
  - Run `mtoc/makeDocumentation.m` to check completeness of your documentation
- Make sure your code is compatible with C++11, `gcc` and `clang`
- when adding new functionality, please also provide test cases (see `tests/cpputest/`)
- Write meaningful commit messages
- Run all tests to ensure nothing got broken
  - Run `tests/cpputest/wrapTestModels.m` followed by CI tests `scripts/run-build.sh && scripts/run-cpputest.sh`
  - Run `examples/amiExamples.m`
- When all tests are passing and you think your code is ready to merge, request a code review

3 SBMLimporter

MATLAB toolbox to generate ODE models from SBML files

4 Model Definition & Simulation

In the following we will give a detailed overview how to specify models in AMIWRAP and how to call the generated simulation files.

4.1 Model Definition

This guide will guide the user on how to specify models in MATLAB. For example implementations see the examples in the example directory.

4.1.1 Header

The model definition needs to be defined as a function which returns a struct with all symbolic definitions and options.

```
function [model] = example_model_syms()
```

4.1.2 Options

Set the options by specifying the respective field of the modelstruct

```
model.(fieldname) = (value)
```

The options specify default options for simulation, parametrisation and compilation. All of these options are optional.

field	description	default
.param	parametrisation 'log'/'log10'/'lin'	'lin'
.debug	flag to compile with debug symbols	false
.forward	flag to activate forward sensitivities	true
.adjoint	flag to activate adjoint sensitivities	true

When set to true, the fields 'noforward' and 'noadjoint' will speed up the time required to compile the model but also disable the respective sensitivity computation.

4.1.3 States

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily.

```
syms state1 state2 state3
```

Create the state vector containing all states:

```
x = [ state1 state2 state3 ];
```

4.1.4 Parameters

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities **will be derived** for all paramaters.

```
syms param1 param2 param3 param4 param5 param6
```

Create the parameters vector

```
p = [ param1 param2 param3 param4 param5 param6 ];
```

4.1.5 Constants

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities with respect to constants **will not be derived**.

```
syms const1 const2
```

Create the parameters vector

```
k = [ const1 const2 ];
```

4.1.6 Differential Equation

For time-dependent differential equations you can specify a symbolic variable for time. This **needs** to be denoted by t.

```
syms t
```

Specify the right hand side of the differential equation f or xdot

```
xdot(1) = [ const1 - param1*state1 ];
xdot(2) = [ +param2*state1 + dirac(t-param3) - const2*state2 ];
xdot(3) = [ param4*state2 ];
```

or

```
f(1) = [ const1 - param1*state1 ];
f(2) = [ +param2*state1 + dirac(t-param3) - const2*state2 ];
f(3) = [ param4*state2 ];
```

The specification of f or xdot may depend on States, Parameters and Constants.

For DAEs also specify the mass matrix.

```
M = [1, 0, 0;...
0, 1, 0;...
0, 0, 0];
```

The specification of M may depend on parameters and constants.

For ODEs the integrator will solve the equation $\dot{x} = f$ and for DAEs the equations $M \cdot \dot{x} = f$. AMICI will decide whether to use CVODES (for ODEs) or IDAS (for DAEs) based on whether the mass matrix is defined or not.

In the definition of the differential equation you can use certain symbolic functions. For a full list of available functions see symbolic_functions.c.

Dirac functions can be used to cause a jump in the respective states at the specified time-point. This is typically used to model injections, or other external stimuli. Spline functions can be used to model time/state dependent response with unkown time/state dependence.

4.1.7 Initial Conditions

Specify the initial conditions. These may depend on Parameters on Constants and must have the same size as x.

```
x0 = [ param4, 0, 0 ];
```

4.1.8 Observables

Specify the observables. These may depend on Parameters and Constants.

```
y(1) = state1 + state2;
y(2) = state3 - state2;
```

In the definition of the observable you can use certain symbolic functions. For a full list of available functions see symbolic_functions.c. Dirac functions in observables will have no effect.

4.1.9 Events

Specifying events is optional. Events are specified in terms of a trigger function, a bolus fuction and an output function. The roots of the trigger function defines the occurences of the event. The bolus function defines the change in the state on event occurences. The output function defines the expression which is evaluated and reported by the simulation routine on every event occurence. The user can create events by constructing a vector of objects of the class amievent.

```
event(1) = amievent(state1 - state2,0,[]);
```

Events may depend on States, Parameters and Constants but **not** on Observables

4.1.10 Standard Deviation

Specifying of standard deviations is optional. It only has an effect when computing adjoint sensitivities. It allows the user to specify standard deviations of experimental data for Observables and Events.

Standard deviaton for observable data is denoted by sigma_y

```
sigma_y(1) = param5;
```

Standard deviaton for event data is denoted by sigma_y

```
sigma_t(1) = param6;
```

Both sigma_y and sigma_t can either be a scalar or of the same dimension as the Observables / Events function. They can depend on time and Parameters but must not depend on the States or Observables. The values provided in sigma_y and sigma_t will only be used if the value in Sigma_Y or Sigma_T in the user-provided data struct is NaN. See Model Simulation for details.

4.1.11 Attach to Model Struct

Eventually all symbolic expressions need to be attached to the model struct.

```
model.sym.x = x;
model.sym.k = k;
model.sym.event = event;
model.sym.xdot = xdot;
% or
model.sym.f = f;
model.sym.M = M; %only for DAEs
model.sym.p = p;
model.sym.x0 = x0;
model.sym.y = y;
model.sym.sigma_y = sigma_y;
model.sym.sigma_t = sigma_t;
```

4.2 Model Compilation

The model can then be compiled by calling amiwrap:

```
amiwrap(modelname,'example_model_syms',dir,o2flag)
```

Here modelname should be a string defining the modelname, dir should be a string containing the path to the directory in which simulation files should be placed and o2flag is a flag indicating whether second order sensitivities should also be compiled. The user should make sure that the previously defined function 'example_model_syms' is in the user path. Alternatively, the user can also call the function 'example_model_syms'

```
[model] = example_model_syms()
```

and subsequently provide the generated struct to amiwrap(), instead of providing the symbolic function:

```
amiwrap(modelname,model,dir,o2flag)
```

In a similar fashion, the user could also generate multiple model and pass them directly to amiwrap() without generating respective model definition scripts.

See also

amiwrap()

4.3 Model Simulation

After the call to amiwrap() two files will be placed in the specified directory. One is a am_*modelname*.mex and the other is simulate_*modelname*.m. The mex file should never be called directly. Instead the MATLAB script, which acts as a wrapper around the .mex simulation file should be used.

The simulate_*modelname*.m itself carries extensive documentation on how to call the function, what it returns and what additional options can be specified. In the following we will give a short overview of possible function calls.

4.3.1 Integration

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the sol.status flag. Negative values indicated failed integration. The states will then be available as sol.x. The observables will then be available as sol.y. The events will then be available as sol.root. If no event occured there will be an event at the end of the considered interval with the final value of the root function stored in sol.rval.

Alternatively the integration call also be called via

```
[status,t,x,y] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the status flag. Negative values indicated failed integration. The states will then be available as x. The observables will then be available as y. No event output will be given.

4.3.2 Forward Sensitivities

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Set the sensitivity computation to forward sensitivities and Integrate:

```
options.sensi = 1;
options.forward = true;
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the sol.status flag. Negative values indicated failed integration. The states will then be available as sol.x, with the derivative with respect to the parameters in sol.sx. The observables will then be available as sol.y, with the derivative with respect to the parameters in sol.sy. The events will then be available as sol.root, with the derivative with respect to the parameters in sol.sroot. If no event occured there will be an event at the end of the considered interval with the final value of the root function stored in sol.rootval, with the derivative with respect to the parameters in sol.srootval

Alternatively the integration call also be called via

```
[status,t,x,y,sx,sy] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the status flag. Negative values indicated failed integration. The states will then be available as x, with derivative with respect to the parameters in sx. The observables will then be available as y, with derivative with respect to the parameters in sy. No event output will be given.

4.3.3 Adjoint Sensitivities

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Set the sensitivity computation to adjoint sensitivities:

```
options.sensi = 1;
options.adjoint = true;
```

Define Experimental Data:

```
D.Y = [NaN(1,2)],ones(length(t)-1,2)];
D.Sigma_Y = [0.1*ones(length(t)-1,2),NaN(1,2)];
D.T = ones(1,1);
D.Sigma_T = NaN;
```

The NaN values in Sigma_Y and Sigma_T will be replaced by the specification in Standard Deviation. Data points with NaN value will be completely ignored.

Generate a constants vector:

```
kappa = ones(2,1);
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The integration status will be indicated by the sol.status flag. Negative values indicated failed integration. The log-likelihood will then be available as sol.llh and the derivative with respect to the parameters in sol.sllh. Notice that for adjoint sensitivities no state, observable and event sensitivities will be available. Yet this approach can be expected to be significantly faster for systems with a large number of parameters.

4.3.4 Steady State Sensitivities

This will compute state sensitivities according to the formula $s_k^x = -\left(\frac{\partial f}{\partial x}\right)^{-1} \frac{\partial f}{\partial \theta_k}$

In the current implementation this formulation does not allow for conservation laws as this would result in a singular Jacobian.

Define a final timepoint t:

```
t = 100
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Set the sensitivity computation to steady state sensitivities:

```
options.sensi = 1;
options.ss = 1;
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The states will then be available as sol.x, with the derivative with respect to the parameters in sol.sx. The observables will then be available as sol.y, with the derivative with respect to the parameters in sol.sy. Notice that for steady state sensitivities no event sensitivities will be available. For the accuracy of the computed derivatives it is essential that the system is sufficiently close to a steady state. This can be checked by examining the right hand side of the system at the final time-point via sol.xdot.

5 Code Organization

In the following we will briefly outline what happens when a model is compiled. For a more detailed description we refer the reader to the documentation of the individual functions.

After specifying a model (see Model Definition) the user will typically compile the model by invoking amiwrap(). amiwrap() first instantiates an object of the class amimodel. The properties of this object are initialised based on the user-defined model. If the o2flag is active, all subsequent computations will also be carried out on the augmented system, which also includes the equations for forward sensitivities. This allows the computation of second order sensitivities in a forward-forward approach. A forward-adjoint approach will be implemented in the future.

The fun fields of this object will then be populated by amimodel::parseModel(). The amimodel::fun field contains all function definitions of type amifun which are required for model compilation. The set of functions to be considered will depend on the user specification of the model fields amimodel::adjoint and amimodel::forward (see Options) as well as the employed solver (CVODES or IDAS, see Differential Equation). For all considered functions amimodel::parseModel() will check their dependencies via amimodel::checkDeps(). These dependencies are a subset of the user-specified fields of amimodel::fun (see Attach to Model Struct). amimodel::parseModel() compares the hashes of all dependencies against the amimodel::HTable of possible previous compilations and will only compute necessary symbolic expressions if changes in these fields occured.

For all functions for which amimodel::fun exists, amimodel::generateC() will generate C files. These files together with their respective header files will be placed in $AMICIDIR/models/*modelname*. amimodel::generateC() will also generate wrapfunctions.h and wrapfunctions.c. These files define and declare model unspecific wrapper functions around model specific functions. This construction allows us to use to build multiple different models against the same simulation routines by linking different realisations of these wrapper functions.

All the generated C functions are subsequently compiled by amimodel::compileC(). For all functions individual object files are created to reduce the computation cost of code optimization. Moreover necessary code from sundials and SuiteSparse is compiled as object files and placed in /models/*mexext*, where mexext stands for the string returned by matlab to the command mexext. The mex simulation file is compiled from amiwrap.c, linked against all object necessary of sundials, SuiteSparse and model specific functions. Depending on the required solver, the compilation will either include cvodewrap.h or idawrap.h. These files implement solver specific realisations of the AMI... functions used in amiwrap.c and amici.c. This allows the use of the same simulation routines for both CVODES and IDAS.

6 Using AMICI-generated code outside Matlab

AMICI (amiwrap.m) translates the model definition into C++ code which is then compiled into a mex file for MATLAB. Advanced users can use this code within stand-online C/C++ applications for use in non-MATLAB environments (e.g. on high performance computing systems). This section will give a short overview over the generated files and provide a brief introduction of how this code can be included in other applications.

Generated model files

amiwrap.m usually write the model source files to ${AMICI_ROOT_DIR}/models/${MODEL_NAME} by default. The content of a model source directory might look something like this (given `MODEL_NAME=model_steadystate`):

```
CMakeLists.txt
hashes.mat
main.cpp
model_steadystate_deltaqB.cpp
model_steadystate_deltaqB.h
model_steadystate_deltaqB_mexa64.md5
model_steadystate_deltaqB.o
[... many more files model_steadystate_*.(cpp|h|md5|o) ]
wrapfunctions.cpp
wrapfunctions.h
wrapfunctions.o
```

Only `*.cpp` and `*.h` files will be needed for the model; `*.o` and `*.md5` are not required.

Running a simulation

The entry function for running an AMICI simulation is getSimulationResults(), declared in amici.h. This function requires all AMICI options and any experimental data. All options that would normally be passed to `simulate_${MODEL_NAME}()` in MATLAB are passed in a UserData struct (see `udata.h` for info). Any experimental data will be passed as ExpData struct (`edata.h`). The simulation results will be returned in a ReturnData struct (see `rdata.h`).

A scaffold for a standalone simulation program is generated in `main.cpp` in the model source directory. This programm shows how to initialize the above-mentioned structs and how to obtain the simulation results.

Compiling and linking

The complete AMICI API is available through amici.h; this is the only header file that needs to be included. (There are some accessor macro definitions available in udata_accessors.h, rdata_accessors.h and edata_accessors.h which provide shortcuts for accessing struct members of UserData, ReturnData, ExpData, respectively. `amici_hdf5.h` provides some functions for reading and writing `HDF5` files).

You need to compile and link `${AMICI_ROOT_DIR}/models/${MODEL_NAME}/*.cpp`, `${AMICI_ROOT_DIR}/src/*.c`
the SUNDIALS and the SUITESPARSE library.

Along with `main.cpp`, a `CMake` file (`CMakeLists.txt`) will be generated automatically. The CMake file shows the abovementioned library dependencies. These files provide a scaffold for a standalone simulation program. The required numerical libraries are shipped with AMICI. To compile them, run `${AMICI_ROOT_DIR}/scripts/run-tests.sh` once. HDF5 libraries and header files need to be installed separately. More information on how to run the compiled program is provided in `main.cpp`. (NOTE: This sample program should compile and link, but will crash most certainly without further problem-specific adaptations.)

7 Hierarchical Index

7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

8 Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

9 File Index

9.1 File List

Here is a list of all documented files with brief descriptions:

10 Class Documentation

10.1 BackwardProblem Class Reference

class to solve backwards problems.

`#include <backwardproblem.h>`

Static Public Member Functions

- static int workBackwardProblem (UserData ∗udata, TempData ∗tdata, ReturnData ∗rdata, Model ∗model)
- static int handleEventB (int iroot, TempData ∗tdata, Model ∗model)
- static int handleDataPointB (int it, ReturnData ∗rdata, TempData ∗tdata, Solver ∗solver, Model ∗model)
- static int updateHeavisideB (int iroot, TempData ∗tdata, int ne)
- static realtype getTnext (realtype ∗troot, int iroot, realtype ∗tdata, int it, Model ∗model)

10.1.1 Detailed Description

solves the backwards problem for adjoint sensitivity analysis and handles events and data-points

Definition at line 19 of file backwardproblem.h.

10.1.2 Member Function Documentation

10.1.2.1 workBackwardProblem()

```
int workBackwardProblem (
            UserData * udata,
            TempData * tdata,
            ReturnData * rdata,
            Model * model ) [static]
```

workBackwardProblem solves the backward problem. if adjoint sensitivities are enabled this will also compute sensitivies workForwardProblem should be called before this is function is called

Parameters

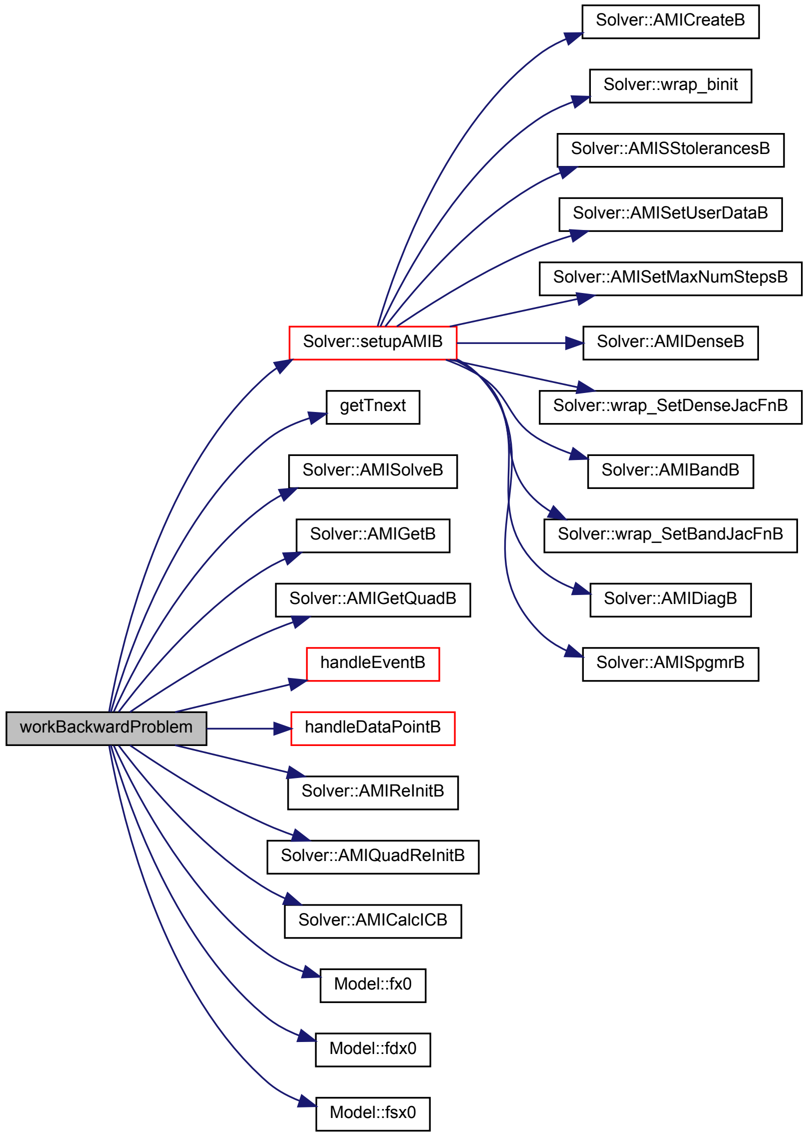
in	*udata*	pointer to the user data struct **Type**: UserData
in	*tdata*	pointer to the temporary data struct **Type**: TempData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*model*	pointer to model specification object **Type**: Model

Returns

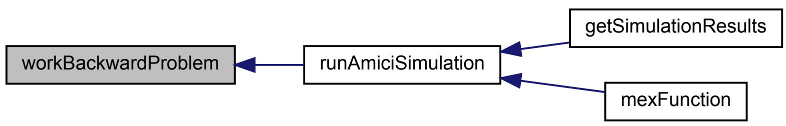
int status flag

Definition at line 10 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.2.2 handleEventB()

```
int handleEventB (
            int iroot,
            TempData * tdata,
            Model * model ) [static]
```

handleEventB executes everything necessary for the handling of events for the backward problem

Parameters

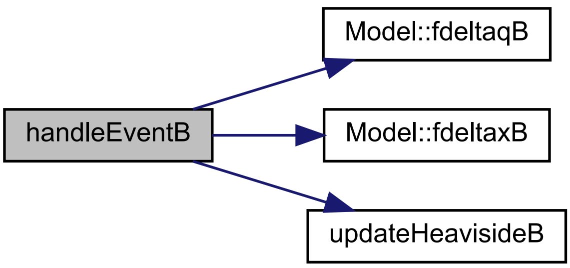
out	*iroot*	index of event **Type**: int
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*model*	pointer to model specification object **Type**: Model

Returns

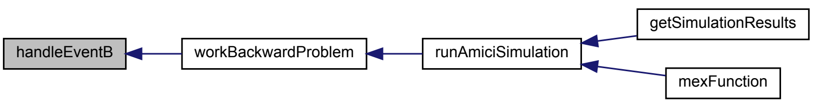
status flag indicating success of execution
Type: int

Definition at line 174 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.2.3 handleDataPointB()

```
int handleDataPointB (
            int it,
            ReturnData * rdata,
            TempData * tdata,
            Solver * solver,
            Model * model ) [static]
```

handleDataPoint executes everything necessary for the handling of data points for the backward problems

Parameters

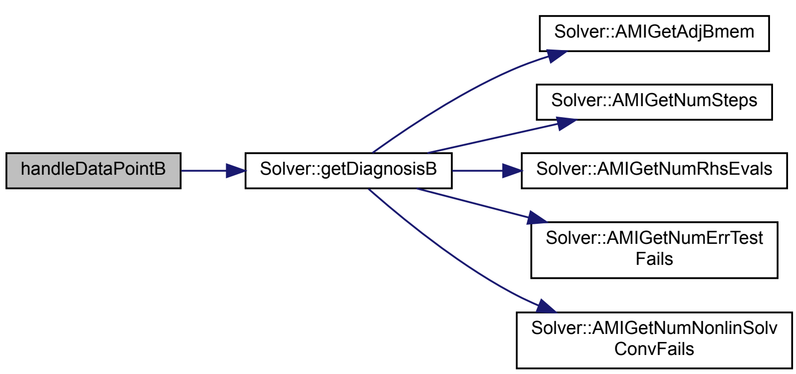
in	*it*	index of data point **Type**: int
out	*rdata*	pointer to the return data struct **Type**: ReturnData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*solver*	pointer to solver object **Type**: Solver
in	*model*	pointer to model specification object **Type**: Model

Returns

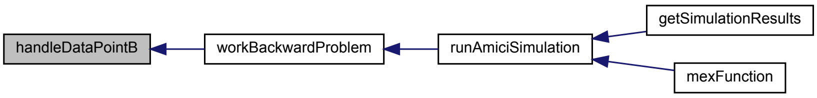
status flag indicating success of execution
Type: int

Definition at line 246 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.1.2.4 updateHeavisideB()

```
int updateHeavisideB (
            int iroot,
            TempData * tdata,
            int ne ) [static]
```

updateHeavisideB updates the heaviside variables h on event occurences for the backward problem

Parameters

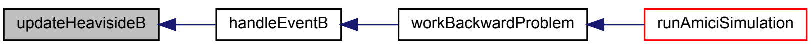
in	*iroot*	discontinuity occurance index **Type**: int
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*ne*	number of events **Type**: int

Returns

status flag indicating success of execution
Type: int

Definition at line 277 of file backwardproblem.cpp.

Here is the caller graph for this function:



10.1.2.5 getTnext()

```
realtype getTnext (
            realtype * troot,
            int iroot,
            realtype * tdata,
            int it,
            Model * model ) [static]
```

getTnext computes the next timepoint to integrate to. This is the maximum of tdata and troot but also takes into account if it<0 or iroot<0 where these expressions do not necessarily make sense

Parameters

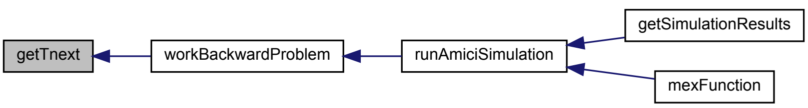
in	*troot*	timepoint of next event **Type**: realtype
in	*iroot*	index of next event **Type**: int
in	*tdata*	timepoint of next data point **Type**: realtype
in	*it*	index of next data point **Type**: int
in	*model*	pointer to model specification object **Type**: Model

Returns

tnext next timepoint
Type: realtype

Definition at line 301 of file backwardproblem.cpp.

Here is the caller graph for this function:



10.2 ExpData Class Reference

struct that carries all information about experimental data

```
#include <edata.h>
```

Public Member Functions

- ExpData ()
- ExpData (const UserData *udata, Model *model)
- void setDefaults ()

Public Attributes

- double ∗ my
- double ∗ sigmay
- double ∗ mz
- double ∗ mrz
- double ∗ sigmaz

10.2.1 Detailed Description

Definition at line 8 of file edata.h.

10.2.2 Constructor & Destructor Documentation

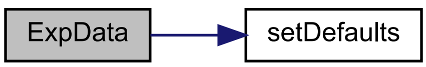
10.2.2.1 ExpData() [1/2]

```
ExpData ( )
```

default constructor

Definition at line 7 of file edata.cpp.

Here is the call graph for this function:



10.2.2.2 ExpData() [2/2]

```
ExpData (
            const UserData * udata,
            Model * model )
```

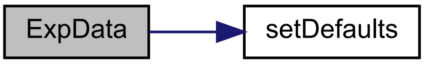
constructor that initializes with UserData and model

Parameters

out	*udata*	pointer to the return data struct **Type**: ReturnData
in	*model*	pointer to model specification object **Type**: Model

Definition at line 9 of file edata.cpp.

Here is the call graph for this function:



10.2.3 Member Function Documentation

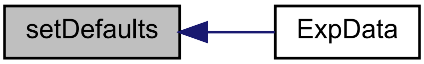
10.2.3.1 setDefaults()

```
void setDefaults ( )
```

initialization with default values

Definition at line 26 of file edata.cpp.

Here is the caller graph for this function:



10.2.4 Member Data Documentation

10.2.4.1 my

```
double* my
```

observed data

Definition at line 20 of file edata.h.

10.2.4.2 sigmay

```
double* sigmay
```

standard deviation of observed data

Definition at line 22 of file edata.h.

10.2.4.3 mz

```
double* mz
```

observed events

Definition at line 25 of file edata.h.

10.2.4.4 mrz

```
double* mrz
```

observed roots

Definition at line 27 of file edata.h.

10.2.4.5 sigmaz

```
double* sigmaz
```

standard deviation of observed events/roots

Definition at line 29 of file edata.h.

10.3 ForwardProblem Class Reference

The ForwardProblem class groups all functions for solving the backwards problem. Has only static members.

```
#include <forwardproblem.h>
```

Static Public Member Functions

- static int workForwardProblem (UserData *udata, TempData *tdata, ReturnData *rdata, const ExpData *edata, Model *model)
- static int handleEvent (realtype *tlastroot, UserData *udata, ReturnData *rdata, const ExpData *edata, TempData *tdata, int seflag, Solver *solver, Model *model)
- static int storeJacobianAndDerivativeInReturnData (TempData *tdata, ReturnData *rdata, Model *model)
- static int getEventOutput (UserData *udata, ReturnData *rdata, const ExpData *edata, TempData *tdata, Model *model)
- static int prepEventSensis (int ie, ReturnData *rdata, const ExpData *edata, TempData *tdata, Model *model)
- static int getEventSensisFSA (int ie, ReturnData *rdata, const ExpData *edata, TempData *tdata, Model *model)
- static int handleDataPoint (int it, UserData *udata, ReturnData *rdata, const ExpData *edata, TempData *tdata, Solver *solver, Model *model)
- static int getDataOutput (int it, UserData *udata, ReturnData *rdata, const ExpData *edata, TempData *tdata, Solver *solver, Model *model)
- static int prepDataSensis (int it, ReturnData *rdata, const ExpData *edata, TempData *tdata, Model *model)
- static int getDataSensisFSA (int it, UserData *udata, ReturnData *rdata, const ExpData *edata, TempData *tdata, Solver *solver, Model *model)
- static int applyEventBolus (TempData *tdata, Model *model)
- static int applyEventSensiBolusFSA (TempData *tdata, Model *model)
- static int updateHeaviside (TempData *tdata, const int ne)

10.3.1 Detailed Description

Definition at line 18 of file forwardproblem.h.

10.3.2 Member Function Documentation

10.3.2.1 workForwardProblem()

```
int workForwardProblem (
            UserData * udata,
            TempData * tdata,
            ReturnData * rdata,
            const ExpData * edata,
            Model * model )  [static]
```

workForwardProblem solves the forward problem. if forward sensitivities are enabled this will also compute sensitivies

Parameters

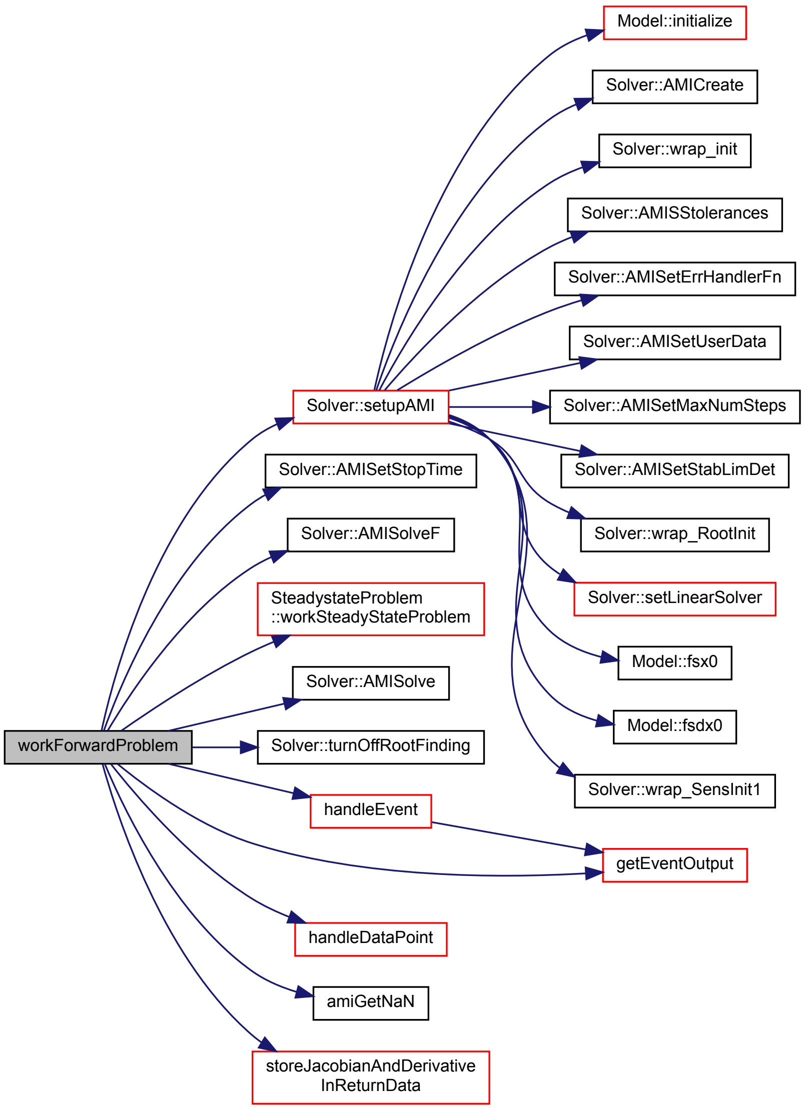
in	*udata*	pointer to the user data struct **Type**: UserData
in	*tdata*	pointer to the temporary data struct **Type**: TempData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
out	*edata*	pointer to the experimental data struct **Type**: ExpData
in	*model*	pointer to model specification object **Type**: Model

Returns

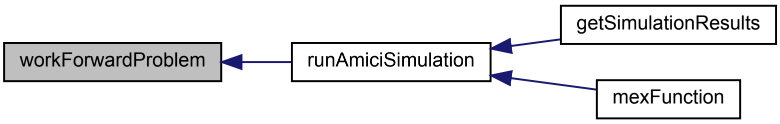
int status flag indicating success of execution
Type: int

Definition at line 11 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.2 handleEvent()

```
int handleEvent (
            realtype * tlastroot,
            UserData * udata,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            int seflag,
            Solver * solver,
            Model * model )  [static]
```

handleEvent executes everything necessary for the handling of events

Parameters

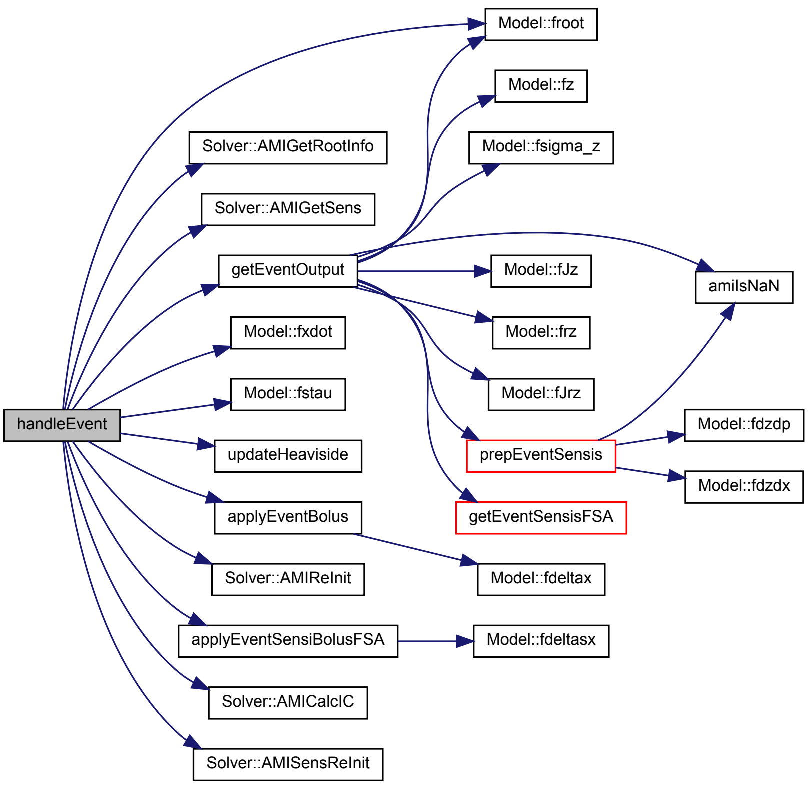
out	*tlastroot*	pointer to the timepoint of the last event **Type**: *realtype
in	*udata*	pointer to the user data struct **Type**: UserData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*seflag*	flag indicating whether this is a secondary event **Type**: int
in	*solver*	pointer to solver object **Type**: Solver
in	*model*	pointer to model specification object **Type**: Model

Returns

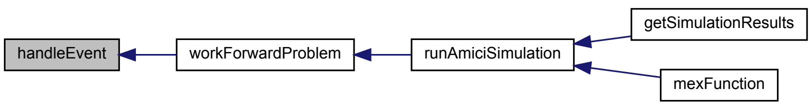
status flag indicating success of execution
Type: int

Definition at line 123 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.3 storeJacobianAndDerivativeInReturnData()

```
int storeJacobianAndDerivativeInReturnData (
            TempData * tdata,
            ReturnData * rdata,
            Model * model )  [static]
```

evalues the Jacobian and differential equation right hand side, stores it in tdata and and copies it to rdata

Parameters

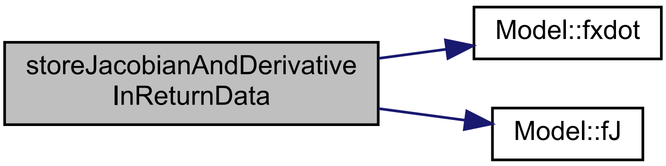
out	*tdata*	pointer to the temporary data struct **Type**: TempData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*model*	pointer to model specification object **Type**: Model

Returns

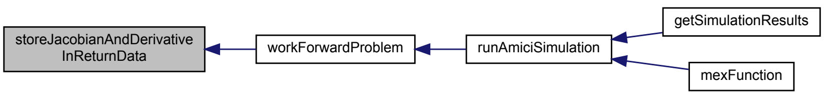
void

Definition at line 321 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.4 getEventOutput()

```
int getEventOutput (
            UserData * udata,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            Model * model ) [static]
```

getEventOutput extracts output information for events

Parameters

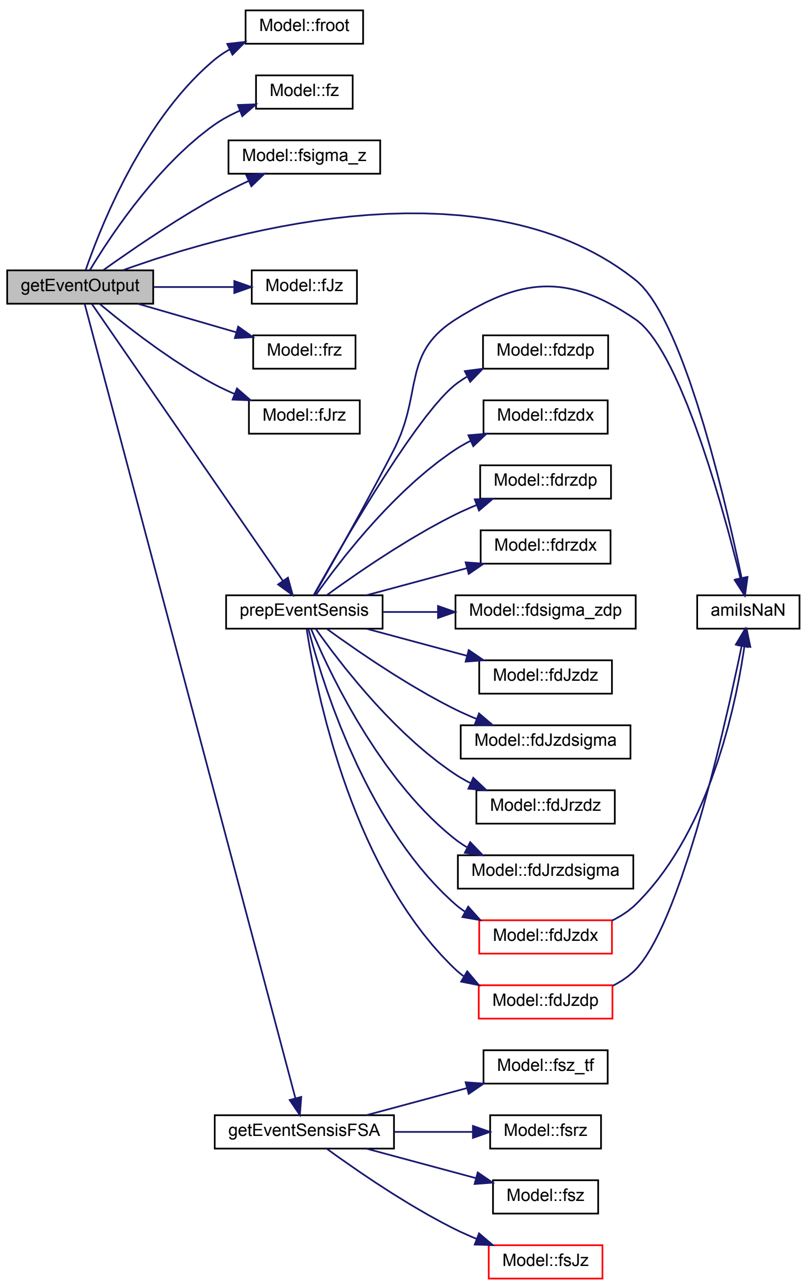
`in`	*udata*	pointer to the user data struct **Type**: UserData
`out`	*rdata*	pointer to the return data struct **Type**: ReturnData
`in`	*edata*	pointer to the experimental data struct **Type**: ExpData
`out`	*tdata*	pointer to the temporary data struct **Type**: TempData
`in`	*model*	pointer to model specification object **Type**: Model

Returns

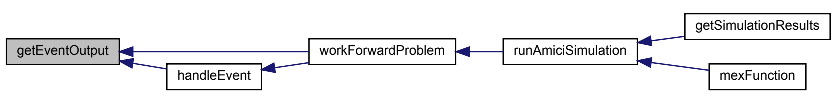
status flag indicating success of execution
Type: int

Definition at line 370 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.5 prepEventSensis()

```
int prepEventSensis (
            int ie,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            Model * model )  [static]
```

prepEventSensis preprocesses the provided experimental data to compute event sensitivities via adjoint or forward methods later on

Parameters

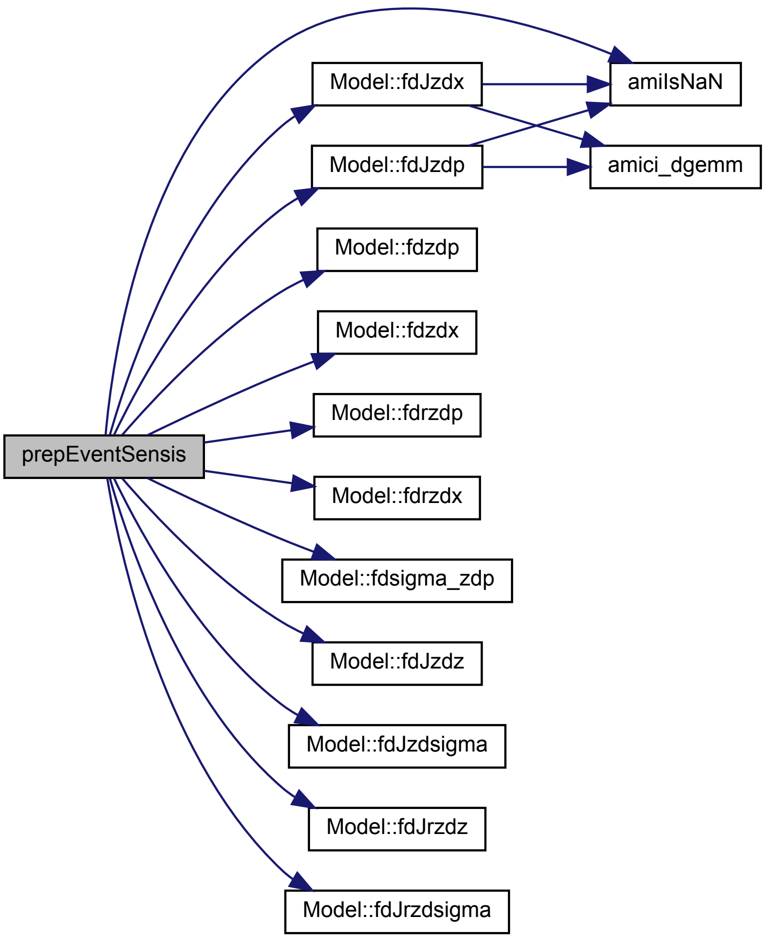
in	*ie*	index of current event **Type**: int
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*model*	pointer to model specification object **Type**: Model

Returns

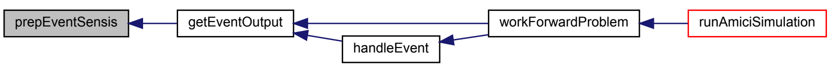
status flag indicating success of execution
Type: int

Definition at line 469 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.6 getEventSensisFSA()

```
int getEventSensisFSA (
            int ie,
            ReturnData * rdata,
            const ExpData * edata,
```

```
TempData * tdata,
Model * model ) [static]
```

getEventSensisFSA extracts event information for forward sensitivity analysis

Parameters

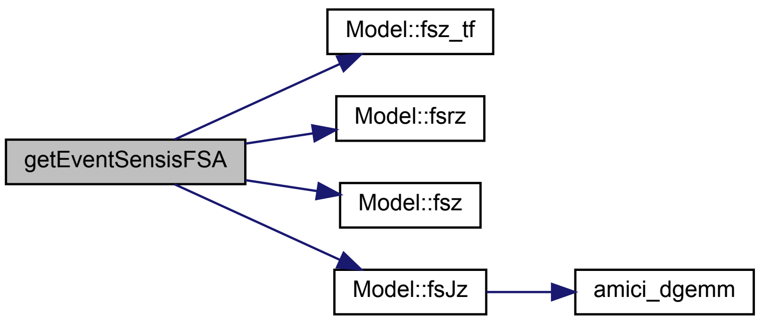
in	*ie*	index of event type **Type**: int
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
in	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*model*	pointer to model specification object **Type**: Model

Returns

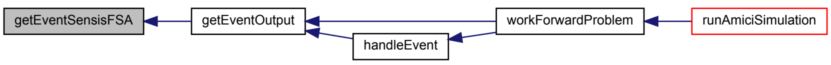
status flag indicating success of execution
Type: int

Definition at line 582 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.7 handleDataPoint()

```
int handleDataPoint (
            int it,
            UserData * udata,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            Solver * solver,
            Model * model )  [static]
```

handleDataPoint executes everything necessary for the handling of data points

Parameters

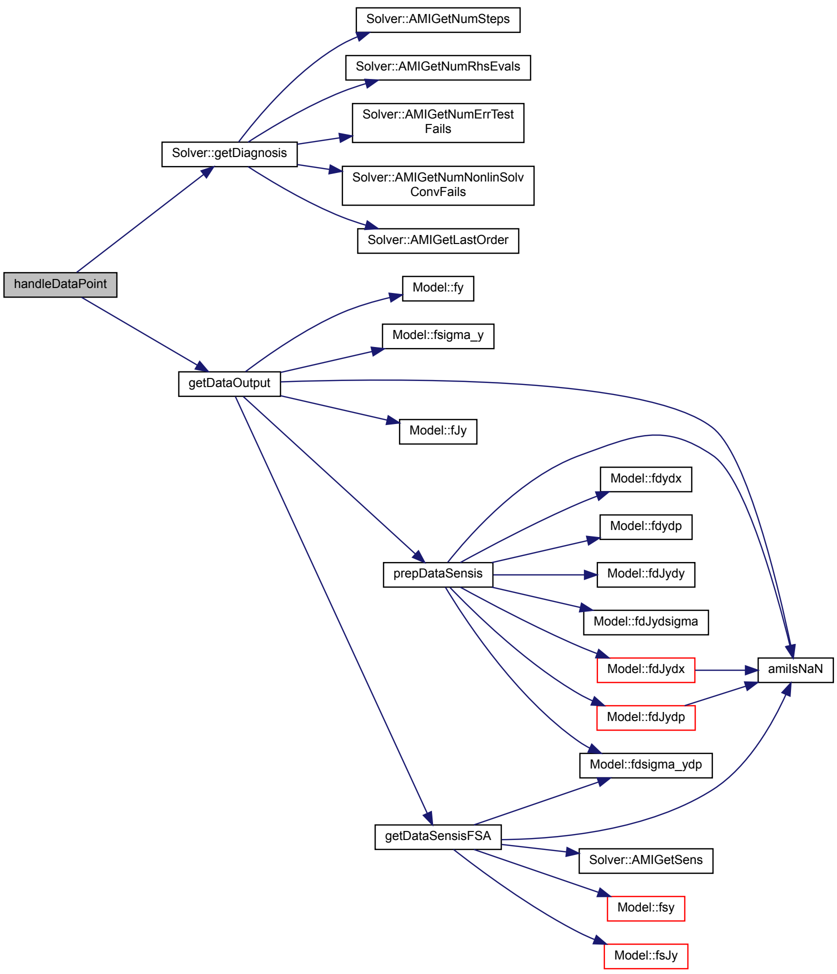
in	*it*	index of data point **Type**: int
in	*udata*	pointer to the user data struct **Type**: UserData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*solver*	pointer to solver object **Type**: Solver
in	*model*	pointer to model specification object **Type**: Model

Returns

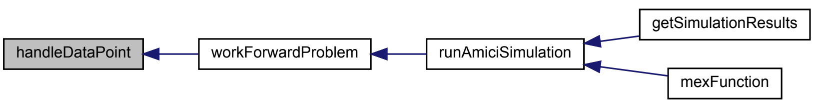
status flag indicating success of execution
Type: int

Definition at line 625 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.8 getDataOutput()

```
int getDataOutput (
            int it,
            UserData * udata,
            ReturnData * rdata,
```

```
const ExpData * edata,
TempData * tdata,
Solver * solver,
Model * model )  [static]
```

getDataOutput extracts output information for data-points

Parameters

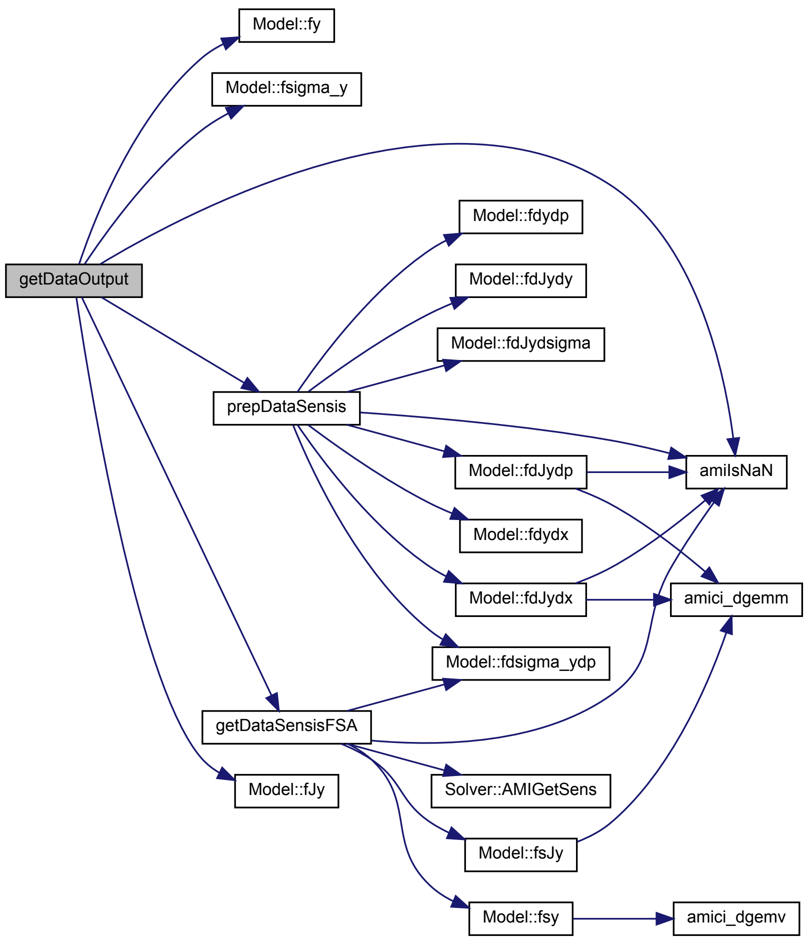
in	*it*	index of current timepoint **Type**: int
in	*udata*	pointer to the user data struct **Type**: UserData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*solver*	pointer to solver object **Type**: Solver
in	*model*	pointer to model specification object **Type**: Model

Returns

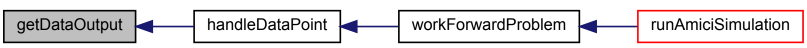
status flag indicating success of execution
Type: int

Definition at line 664 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.9 prepDataSensis()

```
int prepDataSensis (
            int it,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            Model * model ) [static]
```

prepDataSensis preprocesses the provided experimental data to compute sensitivities via adjoint or forward methods later on

Parameters

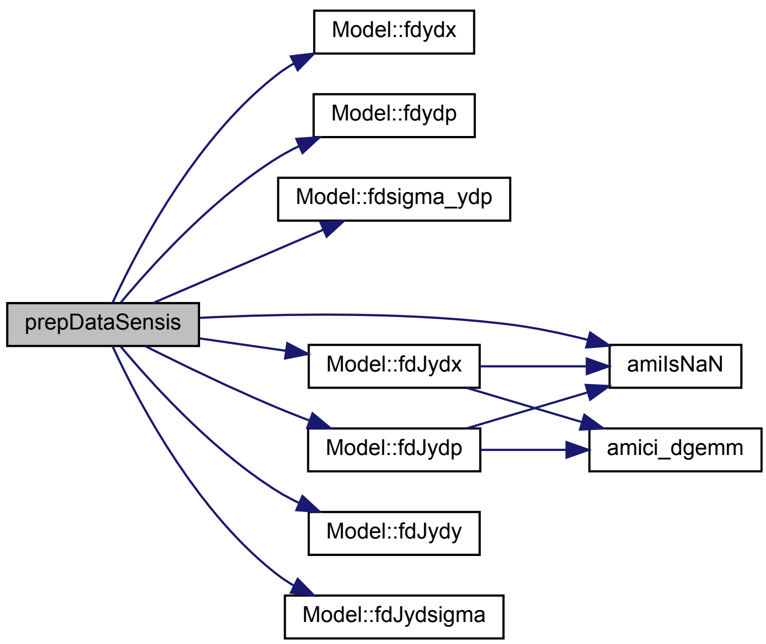
`in`	*it*	index of current timepoint **Type**: int
`out`	*rdata*	pointer to the return data struct **Type**: ReturnData
`in`	*edata*	pointer to the experimental data struct **Type**: ExpData
`out`	*tdata*	pointer to the temporary data struct **Type**: TempData
`in`	*model*	pointer to model specification object **Type**: Model

Returns

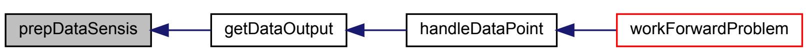
status flag indicating success of execution
Type: int

Definition at line 726 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.10 getDataSensisFSA()

```
int getDataSensisFSA (
            int it,
            UserData * udata,
            ReturnData * rdata,
            const ExpData * edata,
            TempData * tdata,
            Solver * solver,
            Model * model ) [static]
```

getDataSensisFSA extracts data information for forward sensitivity analysis

Parameters

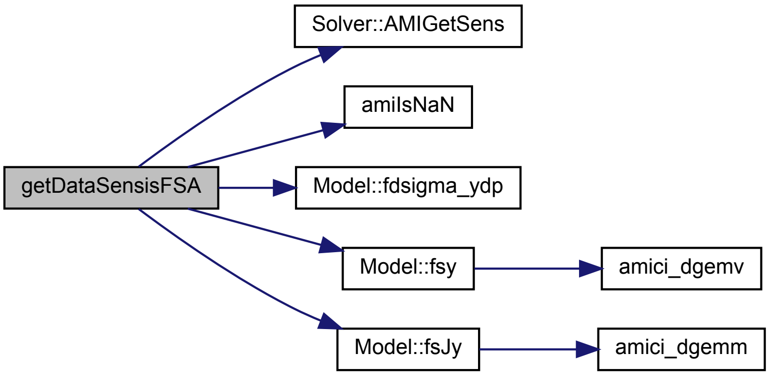
in	*it*	index of current timepoint **Type**: int
in	*udata*	pointer to the user data struct **Type**: UserData
out	*rdata*	pointer to the return data struct **Type**: ReturnData
in	*edata*	pointer to the experimental data struct **Type**: ExpData
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*solver*	pointer to solver object **Type**: Solver
in	*model*	pointer to model specification object **Type**: Model

Returns

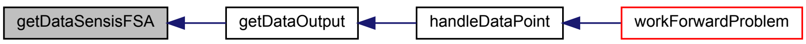
status flag indicating success of execution
Type: int

Definition at line 807 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.3.2.11 applyEventBolus()

```
int applyEventBolus (
            TempData * tdata,
            Model * model ) [static]
```

applyEventBolus applies the event bolus to the current state

Parameters

out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*model*	pointer to model specification object **Type**: Model

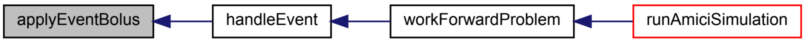
Returns

status flag indicating success of execution
Type: int

Definition at line 881 of file forwardproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



10.3.2.12 applyEventSensiBolusFSA()

```
int applyEventSensiBolusFSA (
            TempData * tdata,
            Model * model ) [static]
```

applyEventSensiBolusFSA applies the event bolus to the current sensitivities

Parameters

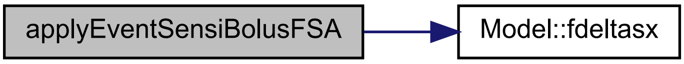
out	*tdata*	pointer to the temporary data struct **Type**: TempData
in	*model*	pointer to model specification object **Type**: Model

Returns

status flag indicating success of execution
Type: int

Definition at line 917 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

10.3.2.13 updateHeaviside()

```
int updateHeaviside (
            TempData * tdata,
            const int ne )  [static]
```

updateHeaviside updates the heaviside variables h on event occurences

Parameters

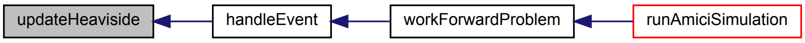
in	*ne*	number of events
out	*tdata*	pointer to the temporary data struct **Type**: TempData

Returns

status = status flag indicating success of execution
Type: int;

Definition at line 955 of file forwardproblem.cpp.

Here is the caller graph for this function:



10.4 Model Class Reference

The Model class represents an AMICI ODE model. The model does not contain any data, its state should never change.

```
#include <amici_model.h>
```

Public Member Functions

- Model ()
- Model (const int np, const int nx, const int nxtrue, const int nk, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const AMICI_o2mode o2mode)
- virtual Solver * getSolver ()
- virtual int fx0 (N_Vector x0, void *user_data)
- virtual int fdx0 (N_Vector x0, N_Vector dx0, void *user_data)
- virtual int fsx0 (N_Vector *sx0, N_Vector x, N_Vector dx, void *user_data)
- virtual int fsdx0 (N_Vector *sdx0, N_Vector x, N_Vector dx, void *user_data)
- virtual int fJ (long int N, realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xdot, DlsMat J, void *user_data, N_Vector tmp1, N_Vector tmp2, N_Vector tmp3)
- virtual int fJB (long int NeqBdot, realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, DlsMat JB, void *user_data, N_Vector tmp1B, N_Vector tmp2B, N_Vector tmp3B)
- virtual int fJDiag (realtype t, N_Vector JDiag, N_Vector x, void *user_data)
- virtual int fJv (N_Vector v, N_Vector Jv, realtype t, N_Vector x, N_Vector xdot, void *user_data, N_Vector tmp)
- virtual int froot (realtype t, N_Vector x, N_Vector dx, realtype *root, void *user_data)
- virtual int frz (realtype t, int ie, N_Vector x, TempData *tdata, ReturnData *rdata)
- virtual int fsrz (realtype t, int ie, N_Vector x, N_Vector *sx, TempData *tdata, ReturnData *rdata)
- virtual int fstau (realtype t, int ie, N_Vector x, N_Vector *sx, TempData *tdata)
- virtual int fy (realtype t, int it, N_Vector x, void *user_data, ReturnData *rdata)
- virtual int fdydp (realtype t, int it, N_Vector x, TempData *tdata)
- virtual int fdydx (realtype t, int it, N_Vector x, TempData *tdata)
- virtual int fz (realtype t, int ie, N_Vector x, TempData *tdata, ReturnData *rdata)
- virtual int fsz (realtype t, int ie, N_Vector x, N_Vector *sx, TempData *tdata, ReturnData *rdata)
- virtual int fdzdp (realtype t, int ie, N_Vector x, TempData *tdata)
- virtual int fdzdx (realtype t, int ie, N_Vector x, TempData *tdata)
- virtual int fdrzdp (realtype t, int ie, N_Vector x, TempData *tdata)
- virtual int fdrzdx (realtype t, int ie, N_Vector x, TempData *tdata)
- virtual int fxdot (realtype t, N_Vector x, N_Vector dx, N_Vector xdot, void *user_data)
- virtual int fxBdot (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector xBdot, void *user_data)
- virtual int fqBdot (realtype t, N_Vector x, N_Vector xB, N_Vector qBdot, void *user_data)
- virtual int fdxdotdp (realtype t, N_Vector x, N_Vector dx, void *user_data)
- virtual int fdeltax (realtype t, int ie, N_Vector x, N_Vector xdot, N_Vector xdot_old, TempData *tdata)
- virtual int fdeltasx (realtype t, int ie, N_Vector x, N_Vector xdot, N_Vector xdot_old, N_Vector *sx, TempData *tdata)
- virtual int fdeltaxB (realtype t, int ie, N_Vector x, N_Vector xB, N_Vector xdot, N_Vector xdot_old, TempData *tdata)
- virtual int fdeltaqB (realtype t, int ie, N_Vector x, N_Vector xB, N_Vector qBdot, N_Vector xdot, N_Vector xdot_old, TempData *tdata)
- virtual int fsigma_y (realtype t, TempData *tdata)
- virtual int fdsigma_ydp (realtype t, TempData *tdata)
- virtual int fsigma_z (realtype t, int ie, TempData *tdata)
- virtual int fdsigma_zdp (realtype t, int ie, TempData *tdata)
- virtual int fJy (realtype t, int it, N_Vector x, TempData *tdata, const ExpData *edata, ReturnData *rdata)

- virtual int fJz (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fJrz (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJydy (realtype t, int it, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJydsigma (realtype t, int it, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJzdz (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJzdsigma (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJrzdz (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fdJrzdsigma (realtype t, int ie, N_Vector x, TempData ∗tdata, const ExpData ∗edata, ReturnData ∗rdata)
- virtual int fsxdot (int Ns, realtype t, N_Vector x, N_Vector xdot, int ip, N_Vector sx, N_Vector sxdot, void ∗user_data, N_Vector tmp1, N_Vector tmp2)
- virtual int fJSparse (realtype t, N_Vector x, N_Vector xdot, SlsMat J, void ∗user_data, N_Vector tmp1, N_Vector tmp2, N_Vector tmp3)
- virtual int fJBand (long int N, long int mupper, long int mlower, realtype t, N_Vector x, N_Vector xdot, DlsMat J, void ∗user_data, N_Vector tmp1, N_Vector tmp2, N_Vector tmp3)
- virtual int fJBandB (long int NeqBdot, long int mupper, long int mlower, realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, DlsMat JB, void ∗user_data, N_Vector tmp1B, N_Vector tmp2B, N_Vector tmp3B)
- virtual int fJvB (N_Vector vB, N_Vector JvB, realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, void ∗user_data, N_Vector tmpB)
- virtual int fJSparseB (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, SlsMat JB, void ∗user_data, N_Vector tmp1B, N_Vector tmp2B, N_Vector tmp3B)
- int fsy (const int it, const TempData ∗tdata, ReturnData ∗rdata)
- int fsz_tf (const int ie, const TempData ∗tdata, ReturnData ∗rdata)
- int fsJy (const int it, const TempData ∗tdata, ReturnData ∗rdata)
- int fdJydp (const int it, TempData ∗tdata, const ExpData ∗edata, const ReturnData ∗rdata)
- int fdJydx (const int it, TempData ∗tdata, const ExpData ∗edata)
- int fsJz (const int ie, TempData ∗tdata, const ReturnData ∗rdata)
- int fdJzdp (const int ie, TempData ∗tdata, const ExpData ∗edata, const ReturnData ∗rdata)
- int fdJzdx (const int ie, TempData ∗tdata, const ExpData ∗edata)
- int initialize (const UserData ∗udata, TempData ∗tdata)
- int initializeStates (const double ∗x0data, TempData ∗tdata)
- int initHeaviside (TempData ∗tdata)

Public Attributes

- const int np
- const int nk
- const int nx
- const int nxtrue
- const int ny
- const int nytrue
- const int nz
- const int nztrue
- const int ne
- const int nw
- const int ndwdx
- const int ndwdp
- const int nnz
- const int nJ
- const int ubw
- const int lbw
- const AMICI_o2mode o2mode
- int ∗ z2event = nullptr
- realtype ∗ idlist = nullptr

10.4.1 Detailed Description

Definition at line 16 of file amici_model.h.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 Model() [1/2]

```
Model ( )
```

default constructor

Definition at line 19 of file amici_model.h.

10.4.2.2 Model() [2/2]

```
Model (
        const int np,
        const int nx,
        const int nxtrue,
        const int nk,
        const int ny,
        const int nytrue,
        const int nz,
        const int nztrue,
        const int ne,
        const int nJ,
        const int nw,
        const int ndwdx,
        const int ndwdp,
        const int nnz,
        const int ubw,
        const int lbw,
        const AMICI_o2mode o2mode )
```

constructor with model dimensions

Parameters

np	number of parameters
nx	number of state variables
nxtrue	number of state variables of the non-augmented model
nk	number of constants
ny	number of observables
nytrue	number of observables of the non-augmented model
nz	number of event observables
nztrue	number of event observables of the non-augmented model
ne	number of events
nJ	number of objective functions

Parameters

nw	number of repeating elements
ndwdx	number of nonzero elements in the x derivative of the repeating elements
ndwdp	number of nonzero elements in the p derivative of the repeating elements
nnz	number of nonzero elements in Jacobian
ubw	upper matrix bandwidth in the Jacobian
lbw	lower matrix bandwidth in the Jacobian
o2mode	second order sensitivity mode

Definition at line 43 of file amici_model.h.

10.4.3 Member Function Documentation

10.4.3.1 getSolver()

```
virtual Solver* getSolver ( )  [virtual]
```

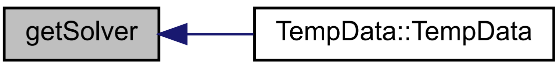
Retrieves the solver object

Returns

Solver solver object
Type: Solver

Definition at line 53 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.2 fx0()

```
virtual int fx0 (
            N_Vector x0,
            void * user_data )  [virtual]
```

Initial states

Parameters

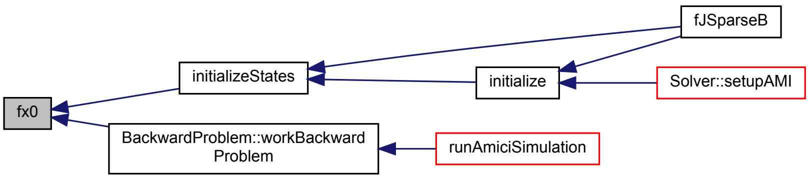
out	*x0*	Vector to which the initial states will be written **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 60 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.3 fdx0()

```
virtual int fdx0 (
            N_Vector x0,
            N_Vector dx0,
            void * user_data ) [virtual]
```

Initial value for time derivative of states (only necessary for DAEs)

Parameters

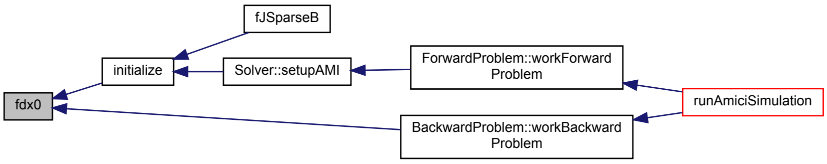
in	*x0*	Vector with the initial states **Type**: N_Vector
out	*dx0*	Vector to which the initial derivative states will be written (only DAE) **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 68 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.4 fsx0()

```
virtual int fsx0 (
            N_Vector * sx0,
            N_Vector x,
            N_Vector dx,
            void * user_data )  [virtual]
```

Initial value for initial state sensitivities

Parameters

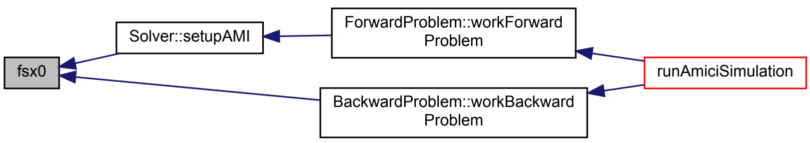
out	*sx0*	Vector to whith the initial state sensitivities **Type**: N_Vector
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 79 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.5 fsdx0()

```
virtual int fsdx0 (
            N_Vector * sdx0,
            N_Vector x,
            N_Vector dx,
            void * user_data ) [virtual]
```

Sensitivity of derivative initial states sensitivities sdx0 (only necessary for DAEs)

Parameters

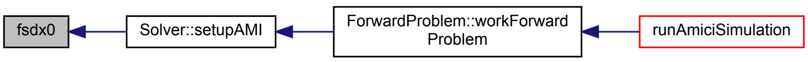
out	*sdx0*	Vector to whith the derivative initial state sensitivities **Type**: N_Vector
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 88 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.6 fJ()

```
virtual int fJ (
            long int N,
            realtype t,
            realtype cj,
            N_Vector x,
            N_Vector dx,
            N_Vector xdot,
            DlsMat J,
            void * user_data,
            N_Vector tmp1,
            N_Vector tmp2,
            N_Vector tmp3 ) [virtual]
```

Jacobian of xdot with respect to states x

Parameters

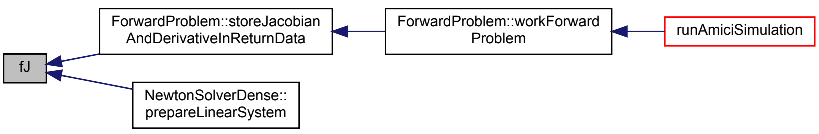
in	*N*	number of state variables **Type**: long_int
in	*t*	timepoint **Type**: realtype
in	*cj*	scaling factor, inverse of the step size (only DAE) **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
out	*J*	Matrix to which the Jacobian will be written **Type**: DlsMat
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1*	temporary storage vector **Type**: N_Vector
in	*tmp2*	temporary storage vector **Type**: N_Vector
in	*tmp3*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 104 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.7 fJB()

```
virtual int fJB (
            long int NeqBdot,
            realtype t,
            N_Vector x,
            N_Vector xB,
```

```
            N_Vector xBdot,
            DlsMat JB,
            void * user_data,
            N_Vector tmp1B,
            N_Vector tmp2B,
            N_Vector tmp3B )  [virtual]
```

Jacobian of xBdot with respect to adjoint state xB

Parameters

in	*NeqBdot*	number of adjoint state variables **Type**: long_int
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*xBdot*	Vector with the adjoint right hand side **Type**: N_Vector
out	*JB*	Matrix to which the Jacobian will be written **Type**: DlsMat
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1B*	temporary storage vector **Type**: N_Vector
in	*tmp2B*	temporary storage vector **Type**: N_Vector
in	*tmp3B*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 119 of file amici_model.h.

10.4.3.8 fJDiag()

```
virtual int fJDiag (
            realtype t,
            N_Vector JDiag,
            N_Vector x,
            void * user_data )  [virtual]
```

diagonalized Jacobian (for preconditioning)

Parameters

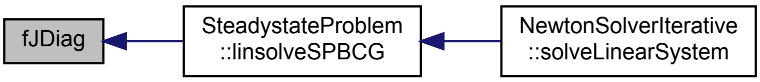
in	*t*	timepoint **Type**: realtype
out	*JDiag*	Vector to which the Jacobian diagonal will be written **Type**: NVector
in	*x*	Vector with the states **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 128 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.9 fJv()

```
virtual int fJv (
            N_Vector v,
            N_Vector Jv,
            realtype t,
            N_Vector x,
            N_Vector xdot,
            void * user_data,
            N_Vector tmp ) [virtual]
```

Matrix vector product of J with a vector v (for iterative solvers)

Parameters

in	*v*	Vector with which the Jacobian is multiplied **Type**: N_Vector
out	*Jv*	Vector to which the Jacobian vector product will be written **Type**: N_Vector
in	*t*	timepoint **Type**: realtype

Parameters

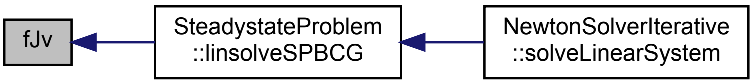
in	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 140 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.10 froot()

```
virtual int froot (
            realtype t,
            N_Vector x,
            N_Vector dx,
            realtype * root,
            void * user_data )  [virtual]
```

Event trigger function for events

Parameters

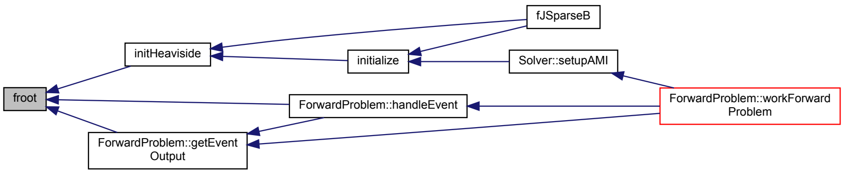
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
out	*root*	array with root function values **Type**: realtype
in	*user_data*	object with model specifications **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 150 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.11 frz()

```
virtual int frz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            ReturnData * rdata ) [virtual]
```

Event root function of events (equal to froot but does not include non-output events)

Parameters

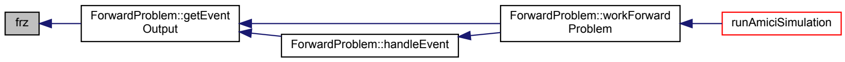
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 160 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.12 fsrz()

```
virtual int fsrz (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector * sx,
            TempData * tdata,
            ReturnData * rdata )  [virtual]
```

Sensitivity of rz, total derivative

Parameters

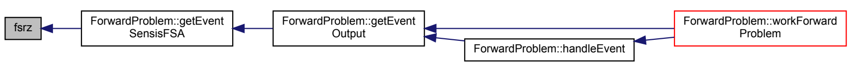
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*sx*	Vector with the state sensitiviies **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 171 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.13 fstau()

```
virtual int fstau (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector * sx,
            TempData * tdata ) [virtual]
```

Sensitivity of event timepoint, total derivative

Parameters

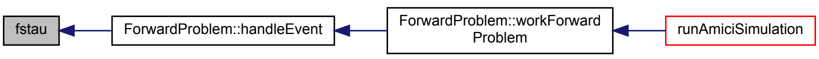
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*sx*	Vector with the state sensitiviies **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 181 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.14 fy()

```
virtual int fy (
            realtype t,
            int it,
            N_Vector x,
            void * user_data,
            ReturnData * rdata ) [virtual]
```

Observables / measurements

Parameters

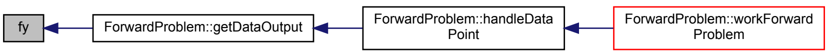
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 191 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.15 fdydp()

```
virtual int fdydp (
            realtype t,
            int it,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of observables y w.r.t. model parameters p

Parameters

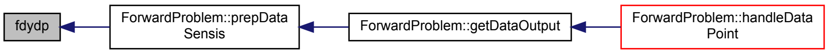
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 200 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.16 fdydx()

```
virtual int fdydx (
            realtype t,
            int it,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of observables y w.r.t. state variables x

Parameters

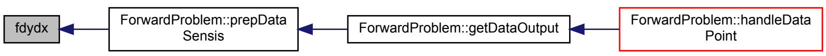
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 209 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.17 fz()

```
virtual int fz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            ReturnData * rdata ) [virtual]
```

Event-resolved measurements

Parameters

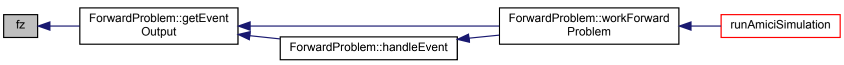
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 219 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.18 fsz()

```
virtual int fsz (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector * sx,
            TempData * tdata,
            ReturnData * rdata ) [virtual]
```

Sensitivity of z, total derivative

Parameters

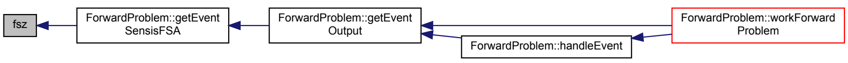
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*sx*	Vector with the state sensitiviies **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 230 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.19 fdzdp()

```
virtual int fdzdp (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of event-resolved measurements z w.r.t. to model parameters p

Parameters

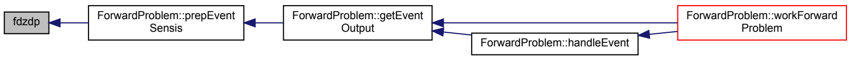
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 239 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.20 fdzdx()

```
virtual int fdzdx (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of event-resolved measurements z w.r.t. to model states x

Parameters

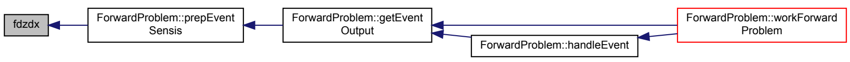
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 248 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.21 fdrzdp()

```
virtual int fdrzdp (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of event-resolved measurements rz w.r.t. to model parameters p

Parameters

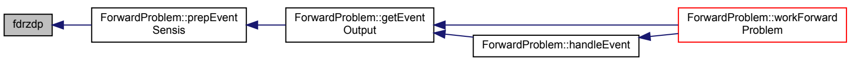
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 257 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.22 fdrzdx()

```
virtual int fdrzdx (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata )  [virtual]
```

Sensitivity of event-resolved measurements rz w.r.t. to model states x

Parameters

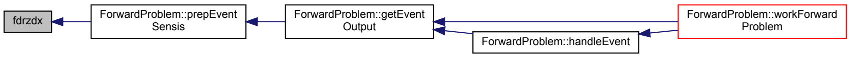
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in,out	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 266 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.23 fxdot()

```
virtual int fxdot (
            realtype t,
            N_Vector x,
            N_Vector dx,
            N_Vector xdot,
            void * user_data )  [virtual]
```

Right hand side of differential equation for states x

Parameters

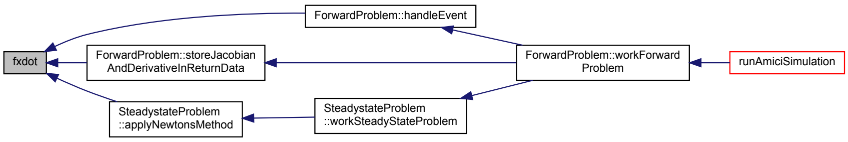
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
out	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 276 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.24 fxBdot()

```
virtual int fxBdot (
            realtype t,
            N_Vector x,
            N_Vector dx,
            N_Vector xB,
            N_Vector dxB,
            N_Vector xBdot,
            void * user_data )  [virtual]
```

Right hand side of differential equation for adjoint state xB

Parameters

in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*dxB*	Vector with the adjoint derivative states (only DAE) **Type**: N_Vector
out	*xBdot*	Vector with the adjoint right hand side **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 288 of file amici_model.h.

10.4.3.25 fqBdot()

```
virtual int fqBdot (
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector qBdot,
            void * user_data ) [virtual]
```

Right hand side of integral equation for quadrature states qB

Parameters

in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
out	*qBdot*	Vector with the adjoint quadrature right hand side **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 298 of file amici_model.h.

10.4.3.26 fdxdotdp()

```
virtual int fdxdotdp (
            realtype t,
            N_Vector x,
            N_Vector dx,
            void * user_data ) [virtual]
```

Sensitivity of dx/dt w.r.t. model parameters p

Parameters

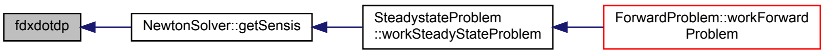
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*dx*	Vector with the derivative states (only DAE) **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 307 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.27 fdeltax()

```
virtual int fdeltax (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector xdot,
            N_Vector xdot_old,
            TempData * tdata )  [virtual]
```

State update functions for events

Parameters

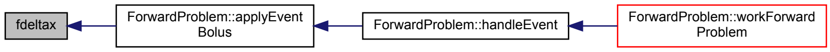
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in,out	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*xdot_old*	Vector with the right hand side before the event **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 318 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.28 fdeltasx()

```
virtual int fdeltasx (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector xdot,
            N_Vector xdot_old,
            N_Vector * sx,
            TempData * tdata )  [virtual]
```

Sensitivity update functions for events, total derivative

Parameters

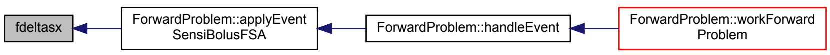
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*xdot_old*	Vector with the right hand side before the event **Type**: N_Vector
in	*sx*	Vector with the state sensitiviies **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 330 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.29 fdeltaxB()

```
virtual int fdeltaxB (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector xB,
            N_Vector xdot,
            N_Vector xdot_old,
            TempData * tdata )  [virtual]
```

Adjoint state update functions for events

Parameters

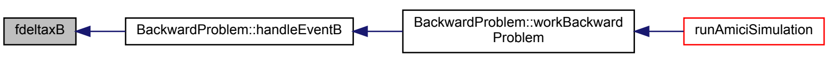
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*xdot_old*	Vector with the right hand side before the event **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 342 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.30 fdeltaqB()

```
virtual int fdeltaqB (
            realtype t,
            int ie,
            N_Vector x,
            N_Vector xB,
            N_Vector qBdot,
            N_Vector xdot,
            N_Vector xdot_old,
            TempData * tdata )  [virtual]
```

Quadrature state update functions for events

Parameters

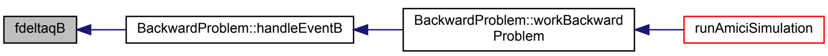
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*qBdot*	Vector with the adjoint quadrature states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*xdot_old*	Vector with the right hand side before the event **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 355 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.31 fsigma_y()

```
virtual int fsigma_y (
            realtype t,
            TempData * tdata )  [virtual]
```

Standard deviation of measurements

Parameters

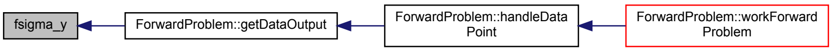
in	*t*	timepoint **Type**: realtype
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 362 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.32 fdsigma_ydp()

```
virtual int fdsigma_ydp (
            realtype t,
            TempData * tdata )  [virtual]
```

Sensitivity of standard deviation of measurements w.r.t. model parameters p

Parameters

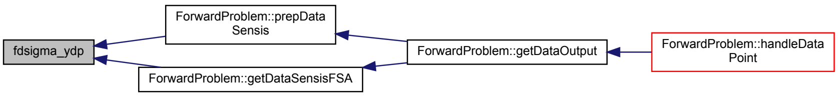
in	*t*	timepoint **Type**: realtype
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 369 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.33 fsigma_z()

```
virtual int fsigma_z (
            realtype t,
            int ie,
            TempData * tdata ) [virtual]
```

Standard deviation of events

Parameters

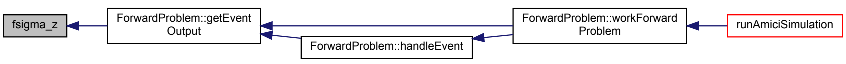
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 377 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.34 fdsigma_zdp()

```
virtual int fdsigma_zdp (
            realtype t,
            int ie,
            TempData * tdata ) [virtual]
```

Sensitivity of standard deviation of events w.r.t. model parameters p

Parameters

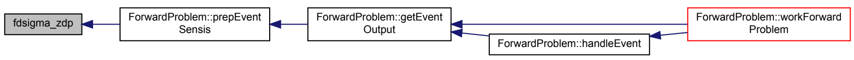
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*tdata*	pointer to temp data object **Type**: TempData

Returns

status flag indicating successful execution
Type: int

Definition at line 385 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.35 fJy()

```
virtual int fJy (
            realtype t,
            int it,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata )  [virtual]
```

negative log-likelihood of time-resolved measurements y

Parameters

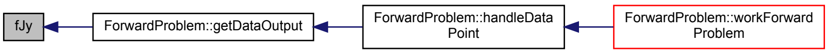
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 396 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.36 fJz()

```
virtual int fJz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata ) [virtual]
```

negative log-likelihood of event-resolved measurements z

Parameters

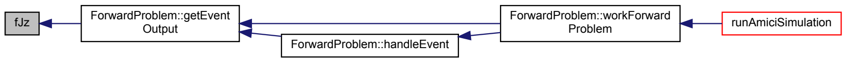
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 407 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.37 fJrz()

```
virtual int fJrz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata ) [virtual]
```

regularization of negative log-likelihood with roots of event-resolved measurements rz

Parameters

in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 418 of file amici_model.h.

Here is the caller graph for this function:

10.4.3.38 fdJydy()

```
virtual int fdJydy (
            realtype t,
            int it,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata ) [virtual]
```

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. observables y

Parameters

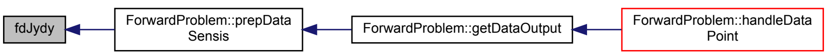
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 429 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.39 fdJydsigma()

```
virtual int fdJydsigma (
            realtype t,
            int it,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata ) [virtual]
```

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigma

Parameters

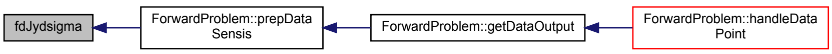
in	*t*	timepoint **Type**: realtype
in	*it*	timepoint index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 440 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.40 fdJzdz()

```
virtual int fdJzdz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata )  [virtual]
```

Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. event observables z

Parameters

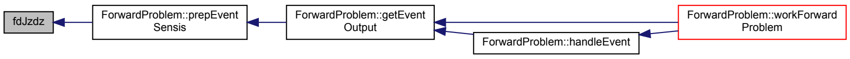
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 451 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.41 fdJzdsigma()

```
virtual int fdJzdsigma (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata )  [virtual]
```

Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. standard deviation sigma

Parameters

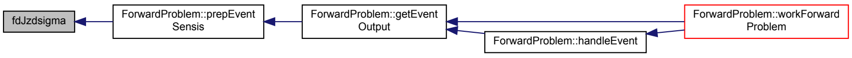
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 462 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.42 fdJrzdz()

```
virtual int fdJrzdz (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata )  [virtual]
```

Sensitivity of event-resolved measurement negative log-likelihood regularization Jrz w.r.t. event observables z

Parameters

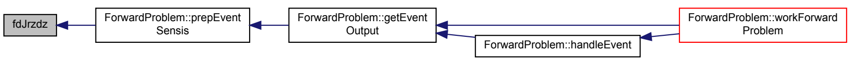
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 473 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.43 fdJrzdsigma()

```
virtual int fdJrzdsigma (
            realtype t,
            int ie,
            N_Vector x,
            TempData * tdata,
            const ExpData * edata,
            ReturnData * rdata ) [virtual]
```

Sensitivity of event-resolved measurement negative log-likelihood regularization Jrz w.r.t. standard deviation sigma

Parameters

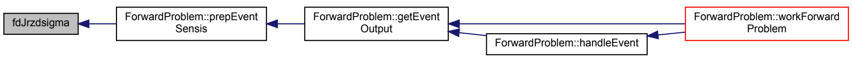
in	*t*	timepoint **Type**: realtype
in	*ie*	event index **Type**: int
in	*x*	Vector with the states **Type**: N_Vector
in	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 484 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.44 fsxdot()

```
virtual int fsxdot (
            int Ns,
            realtype t,
            N_Vector x,
            N_Vector xdot,
            int ip,
            N_Vector sx,
```

```
            N_Vector sxdot,
            void * user_data,
            N_Vector tmp1,
            N_Vector tmp2 )  [virtual]
```

Right hand side of differential equation for state sensitivities sx

Parameters

in	*Ns*	number of parameters **Type**: int
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
in	*ip*	parameter index **Type**: int
in	*sx*	Vector with the state sensitivities **Type**: N_Vector
in	*sxdot*	Vector with the sensitivity right hand side **Type**: N_Vector
in	*user_data*	pointer to temp data object **Type**: TempData
in	*tmp1*	temporary storage vector **Type**: N_Vector
in	*tmp2*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 499 of file amici_model.h.

10.4.3.45 fJSparse()

```
virtual int fJSparse (
            realtype t,
            N_Vector x,
            N_Vector xdot,
            SlsMat J,
            void * user_data,
            N_Vector tmp1,
            N_Vector tmp2,
            N_Vector tmp3 )  [virtual]
```

J in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

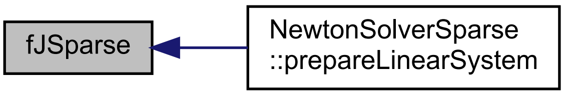
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
out	*J*	Matrix to which the Jacobian will be written **Type**: SlsMat
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1*	temporary storage vector **Type**: N_Vector
in	*tmp2*	temporary storage vector **Type**: N_Vector
in	*tmp3*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 512 of file amici_model.h.

Here is the caller graph for this function:



10.4.3.46 fJBand()

```
virtual int fJBand (
            long int N,
            long int mupper,
            long int mlower,
            realtype t,
            N_Vector x,
            N_Vector xdot,
            DlsMat J,
            void * user_data,
            N_Vector tmp1,
            N_Vector tmp2,
            N_Vector tmp3 )  [virtual]
```

J in banded form (for banded solvers)

Parameters

in	*N*	number of states **Type**: long int
in	*mupper*	upper matrix bandwidth **Type**: long int
in	*mlower*	lower matrix bandwidth **Type**: long int
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xdot*	Vector with the right hand side **Type**: N_Vector
out	*J*	Matrix to which the Jacobian will be written **Type**: DlsMat
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1*	temporary storage vector **Type**: N_Vector
in	*tmp2*	temporary storage vector **Type**: N_Vector
in	*tmp3*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 528 of file amici_model.h.

10.4.3.47 fJBandB()

```
virtual int fJBandB (
            long int NeqBdot,
            long int mupper,
            long int mlower,
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector xBdot,
            DlsMat JB,
            void * user_data,
            N_Vector tmp1B,
            N_Vector tmp2B,
            N_Vector tmp3B )  [virtual]
```

JB in banded form (for banded solvers)

Parameters

in	*NeqBdot*	number of states **Type**: long int
in	*mupper*	upper matrix bandwidth **Type**: long int
in	*mlower*	lower matrix bandwidth **Type**: long int
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*xBdot*	Vector with the adjoint right hand side **Type**: N_Vector
out	*JB*	Matrix to which the Jacobian will be written **Type**: DlsMat
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1B*	temporary storage vector **Type**: N_Vector
in	*tmp2B*	temporary storage vector **Type**: N_Vector
in	*tmp3B*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 545 of file amici_model.h.

10.4.3.48 fJvB()

```
virtual int fJvB (
            N_Vector vB,
            N_Vector JvB,
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector xBdot,
            void * user_data,
            N_Vector tmpB ) [virtual]
```

Matrix vector product of JB with a vector v (for iterative solvers)

Parameters

in	*vB*	Vector with which the Jacobian is multiplied **Type**: N_Vector

Parameters

out	*JvB*	Vector to which the Jacobian vector product will be written **Type**: N_Vector
in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*xBdot*	Vector with the adjoint right hand side **Type**: N_Vector
in	*user_data*	object with model specifications **Type**: TempData
in	*tmpB*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 558 of file amici_model.h.

10.4.3.49 fJSparseB()

```
virtual int fJSparseB (
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector xBdot,
            SlsMat JB,
            void * user_data,
            N_Vector tmp1B,
            N_Vector tmp2B,
            N_Vector tmp3B ) [virtual]
```

JB in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

in	*t*	timepoint **Type**: realtype
in	*x*	Vector with the states **Type**: N_Vector
in	*xB*	Vector with the adjoint states **Type**: N_Vector
in	*xBdot*	Vector with the adjoint right hand side **Type**: N_Vector
out	*JB*	Matrix to which the Jacobian will be written **Type**: DlsMat

Parameters

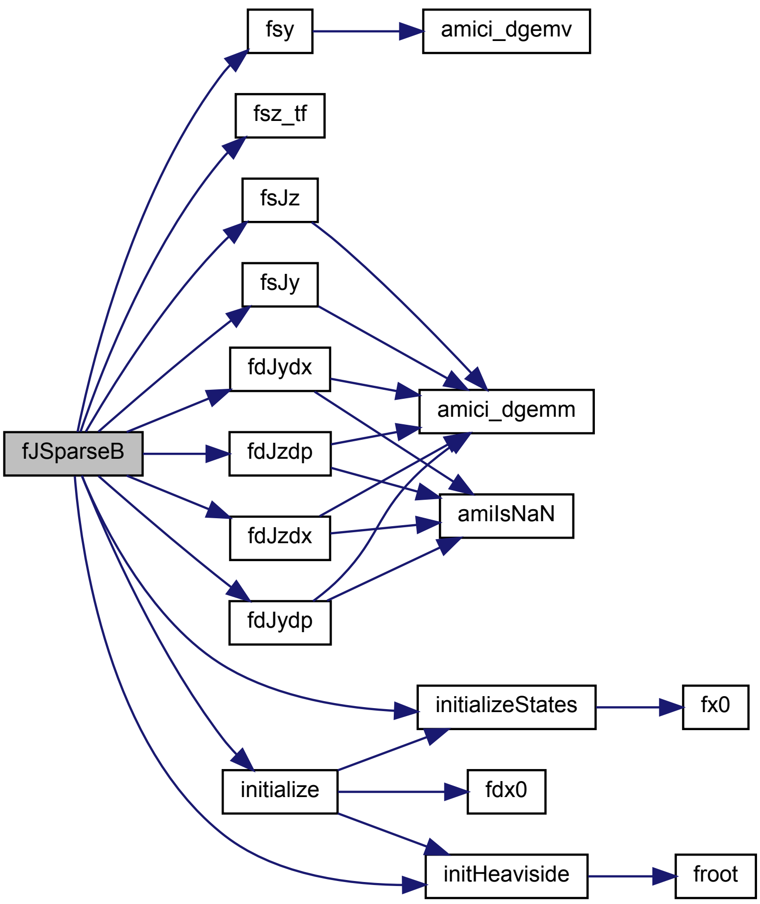
in	*user_data*	object with model specifications **Type**: TempData
in	*tmp1B*	temporary storage vector **Type**: N_Vector
in	*tmp2B*	temporary storage vector **Type**: N_Vector
in	*tmp3B*	temporary storage vector **Type**: N_Vector

Returns

status flag indicating successful execution
Type: int

Definition at line 572 of file amici_model.h.

Here is the call graph for this function:



10.4.3.50 fsy()

```
int fsy (
            const int it,
            const TempData * tdata,
            ReturnData * rdata )
```

Sensitivity of measurements y, total derivative

Parameters

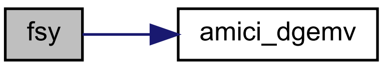
in	*it*	timepoint index **Type**: int
in	*tdata*	pointer to temp data object **Type**: TempData
in,out	*rdata*	pointer to return data object **Type**: ReturnData

Returns

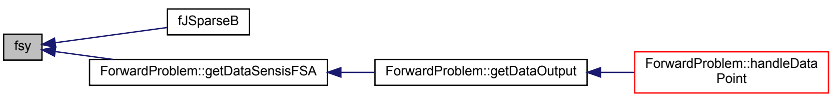
status flag indicating successful execution
Type: int

Definition at line 30 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.51 fsz_tf()

```
int fsz_tf (
            const int ie,
            const TempData * tdata,
            ReturnData * rdata )
```

Sensitivity of z at final timepoint (ignores sensitivity of timepoint), total derivative

Parameters

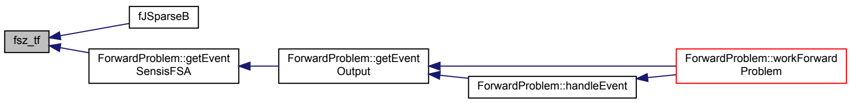
`in`	*ie*	event index **Type**: int
`in`	*tdata*	pointer to temp data object **Type**: TempData
`in,out`	*rdata*	pointer to return data object **Type**: ReturnData

Returns

status flag indicating successful execution
Type: int

Definition at line 58 of file amici_model.cpp.

Here is the caller graph for this function:



10.4.3.52 fsJy()

```
int fsJy (
        const int it,
        const TempData * tdata,
        ReturnData * rdata )
```

Sensitivity of time-resolved measurement negative log-likelihood Jy, total derivative

Parameters

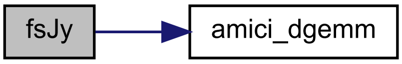
`in`	*it*	timepoint index **Type**: int
`in`	*tdata*	pointer to temp data object **Type**: TempData
`in,out`	*rdata*	pointer to return data object **Type**: ReturnData

Returns

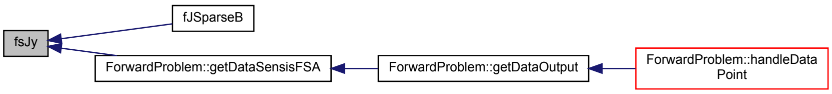
status flag indicating successful execution
Type: int

Definition at line 79 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.53 fdJydp()

```
int fdJydp (
        const int it,
        TempData * tdata,
        const ExpData * edata,
        const ReturnData * rdata )
```

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. parameters

Parameters

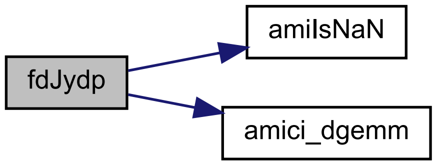
in	*it*	timepoint index **Type**: int
in,out	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in	*rdata*	pointer to return data object **Type**: ReturnData

Returns

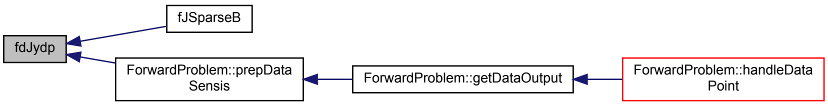
status flag indicating successful execution
Type: int

Definition at line 132 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.54 fdJydx()

```
int fdJydx (
            const int it,
            TempData * tdata,
            const ExpData * edata )
```

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. state variables

Parameters

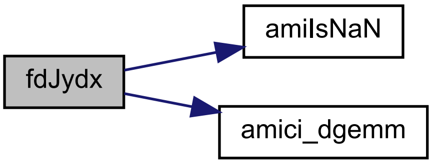
in	*it*	timepoint index **Type**: int
in,out	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData

Returns

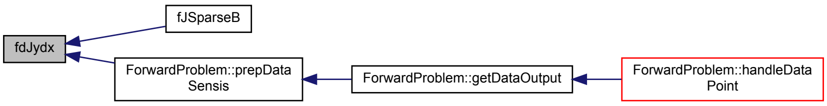
status flag indicating successful execution
Type: int

Definition at line 183 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.55 fsJz()

```
int fsJz (
            const int ie,
            TempData * tdata,
            const ReturnData * rdata )
```

Sensitivity of event-resolved measurement negative log-likelihood Jz, total derivative

Parameters

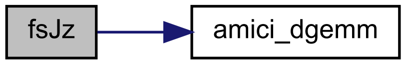
in	*ie*	event index **Type**: int
in,out	*tdata*	pointer to temp data object **Type**: TempData
in	*rdata*	pointer to return data object **Type**: ReturnData

Returns

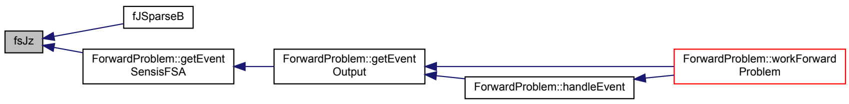
status flag indicating successful execution
Type: int

Definition at line 225 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.56 fdJzdp()

```
int fdJzdp (
        const int ie,
        TempData * tdata,
        const ExpData * edata,
        const ReturnData * rdata )
```

Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. parameters

Parameters

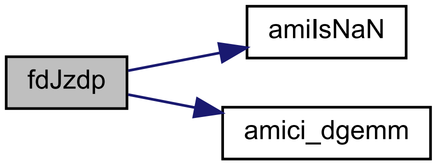
in	*ie*	event index **Type**: int
in,out	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData
in	*rdata*	pointer to return data object **Type**: ReturnData

Returns

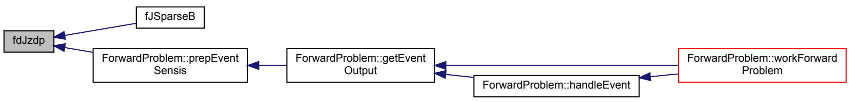
status flag indicating successful execution
Type: int

Definition at line 285 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.57 fdJzdx()

```
int fdJzdx (
            const int ie,
            TempData * tdata,
            const ExpData * edata )
```

Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. state variables

Parameters

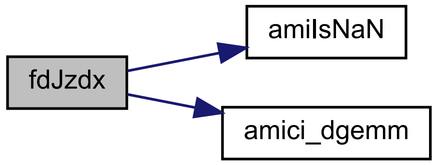
in	*ie*	event index **Type**: int
in,out	*tdata*	pointer to temp data object **Type**: TempData
in	*edata*	pointer to experimental data object **Type**: ExpData

Returns

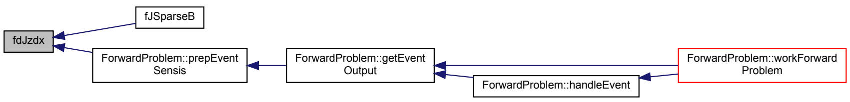
status flag indicating successful execution
Type: int

Definition at line 362 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.58 initialize()

```
int initialize (
            const UserData * udata,
            TempData * tdata )
```

initialization of model properties

Parameters

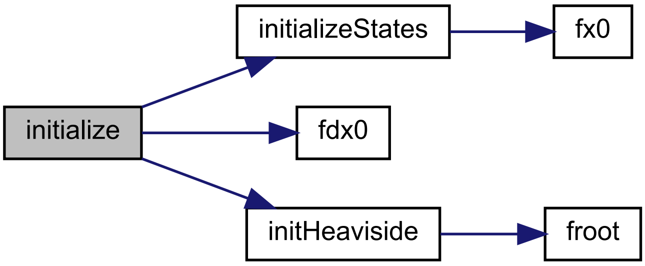
in	*udata*	pointer to user data object **Type**: UserData
out	*tdata*	pointer to temp data object **Type**: TempData

Returns

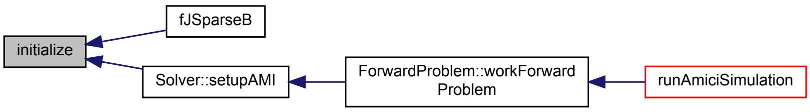
status flag indicating success of execution
Type: int

Definition at line 417 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.59 initializeStates()

```
int initializeStates (
            const double * x0data,
            TempData * tdata )
```

initialization of initial states

Parameters

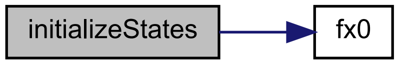
in	*x0data*	array with initial state values **Type**: double
out	*tdata*	pointer to temp data object **Type**: TempData

Returns

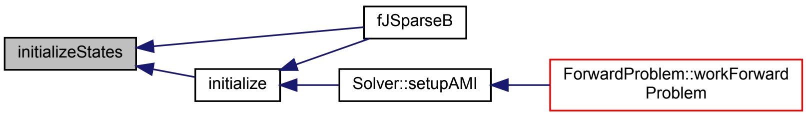
status flag indicating success of execution
Type: int

Definition at line 440 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.3.60 initHeaviside()

```
int initHeaviside (
            TempData * tdata )
```

initHeaviside initialises the heaviside variables h at the intial time t0 heaviside variables activate/deactivate on event occurences

Parameters

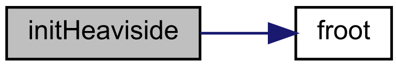
out	*tdata*	pointer to the temporary data struct **Type**: TempData

Returns

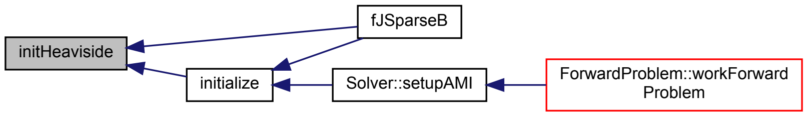
status flag indicating success of execution
Type: int

Definition at line 470 of file amici_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.4.4 Member Data Documentation

10.4.4.1 np

```
const int np
```

total number of model parameters

Definition at line 601 of file amici_model.h.

10.4.4.2 nk

```
const int nk
```

number of fixed parameters

Definition at line 603 of file amici_model.h.

10.4.4.3 nx

```
const int nx
```

number of states

Definition at line 605 of file amici_model.h.

10.4.4.4 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 607 of file amici_model.h.

10.4.4.5 ny

```
const int ny
```

number of observables

Definition at line 609 of file amici_model.h.

10.4.4.6 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 611 of file amici_model.h.

10.4.4.7 nz

```
const int nz
```

number of event outputs

Definition at line 613 of file amici_model.h.

10.4.4.8 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 615 of file amici_model.h.

10.4.4.9 ne

```
const int ne
```

number of events

Definition at line 617 of file amici_model.h.

10.4.4.10 nw

```
const int nw
```

number of common expressions

Definition at line 619 of file amici_model.h.

10.4.4.11 ndwdx

```
const int ndwdx
```

number of derivatives of common expressions wrt x

Definition at line 621 of file amici_model.h.

10.4.4.12 ndwdp

```
const int ndwdp
```

number of derivatives of common expressions wrt p

Definition at line 623 of file amici_model.h.

10.4.4.13 nnz

```
const int nnz
```

number of nonzero entries in jacobian

Definition at line 625 of file amici_model.h.

10.4.4.14 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 627 of file amici_model.h.

10.4.4.15 ubw

```
const int ubw
```

upper bandwith of the jacobian

Definition at line 629 of file amici_model.h.

10.4.4.16 lbw

```
const int lbw
```

lower bandwith of the jacobian

Definition at line 631 of file amici_model.h.

10.4.4.17 o2mode

```
const AMICI_o2mode o2mode
```

flag indicating whether for sensi == AMICI_SENSI_ORDER_SECOND directional or full second order derivative will be computed

Definition at line 634 of file amici_model.h.

10.4.4.18 z2event

```
int* z2event = nullptr
```

index indicating to which event an event output belongs

Definition at line 636 of file amici_model.h.

10.4.4.19 idlist

```
realtype* idlist = nullptr
```

flag array for DAE equations

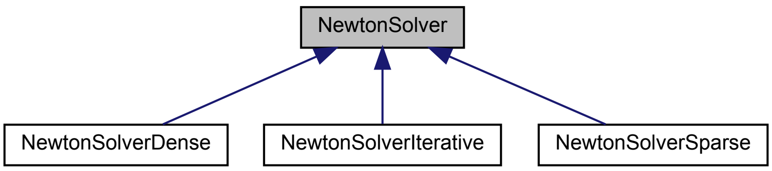
Definition at line 638 of file amici_model.h.

10.5 NewtonSolver Class Reference

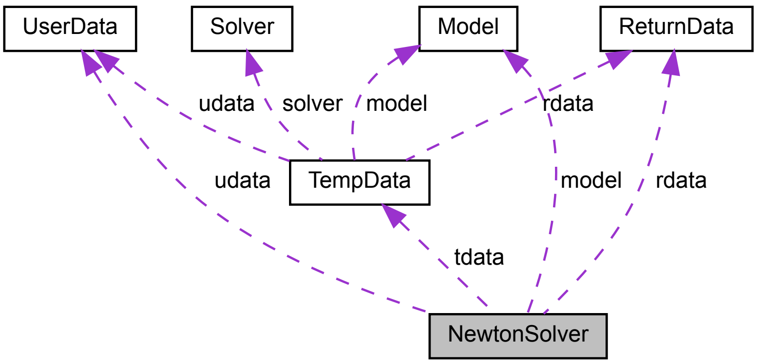
The NewtonSolver class sets up the linear solver for the Newton method.

```
#include <newton_solver.h>
```

Inheritance diagram for NewtonSolver:



Collaboration diagram for NewtonSolver:



Public Member Functions

- NewtonSolver (Model ∗model, ReturnData ∗rdata, UserData ∗udata, TempData ∗tdata)
- int getStep (int ntry, int nnewt, N_Vector delta)
- int getSensis (int it)
- virtual int prepareLinearSystem (int ntry, int nnewt)=0
- virtual int solveLinearSystem (N_Vector rhs)=0

Static Public Member Functions

- static NewtonSolver ∗ getSolver (int linsolType, Model ∗model, ReturnData ∗rdata, UserData ∗udata, TempData ∗tdata, int ∗status)

Protected Attributes

- Model ∗ model
- ReturnData ∗ rdata
- UserData ∗ udata
- TempData ∗ tdata

10.5.1 Detailed Description

Definition at line 21 of file newton_solver.h.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 NewtonSolver()

```
NewtonSolver (
            Model * model,
            ReturnData * rdata,
            UserData * udata,
            TempData * tdata )
```

default constructor, initializes all members with the provided objects

Parameters

in	*model*	pointer to the AMICI model object **Type**: Model
in	*rdata*	pointer to the return data object **Type**: ReturnData
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData

Definition at line 15 of file newton_solver.cpp.

10.5.3 Member Function Documentation

10.5.3.1 getSolver()

```
NewtonSolver * getSolver (
            int linsolType,
            Model * model,
            ReturnData * rdata,
            UserData * udata,
            TempData * tdata,
            int * status ) [static]
```

Tries to determine the steady state of the ODE system by a Newton solver, uses forward intergration, if the Newton solver fails, restarts Newton solver, if integration fails. Computes steady state sensitivities

Parameters

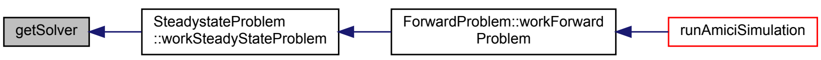
in	*linsolType*	integer indicating which linear solver to use
in	*model*	pointer to the AMICI model object **Type**: Model
in	*udata*	pointer to the user data object **Type**: UserData
in,out	*tdata*	pointer to the temporary data object **Type**: TempData
out	*rdata*	pointer to the return data object **Type**: ReturnData
out	*status*	pointer to integer with flag for success of initialization

Returns

solver NewtonSolver according to the specified linsolType

Definition at line 29 of file newton_solver.cpp.

Here is the caller graph for this function:



10.5.3.2 getStep()

```
int getStep (
            int ntry,
            int nnewt,
            N_Vector delta )
```

Computes the solution of one Newton iteration

Parameters

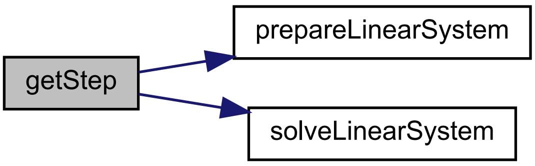
in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step
in,out	*delta*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

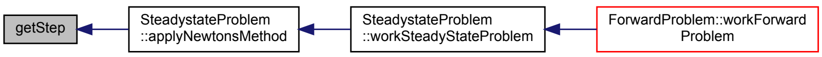
stats integer flag indicating success of the method

Definition at line 99 of file newton_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.5.3.3 getSensis()

```
int getSensis (
        int it )
```

Computes steady state sensitivities

Parameters

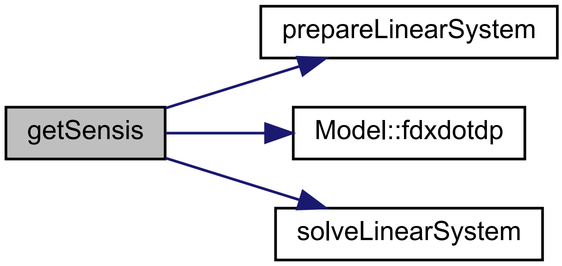
in	*it*	integer index of current time step

Returns

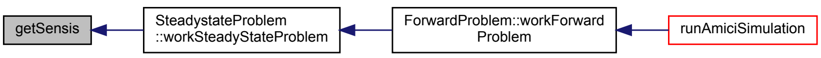
stats integer flag indicating success of the method

Definition at line 121 of file newton_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.5.3.4 prepareLinearSystem()

```
virtual int prepareLinearSystem (
            int ntry,
            int nnewt ) [pure virtual]
```

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

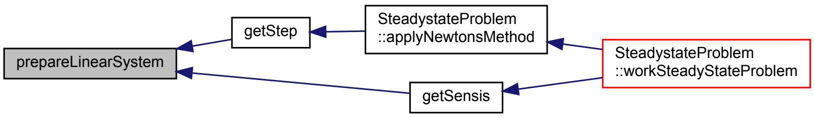
in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step

Returns

stats integer flag indicating success of the method

Implemented in NewtonSolverIterative, NewtonSolverSparse, and NewtonSolverDense.

Here is the caller graph for this function:



10.5.3.5 solveLinearSystem()

```
virtual int solveLinearSystem (
            N_Vector rhs ) [pure virtual]
```

Solves the linear system for the Newton step

Parameters

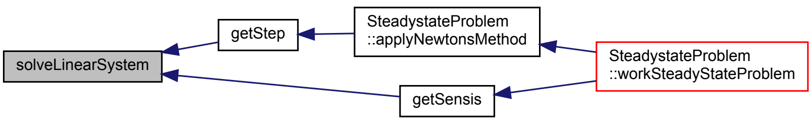
in,out	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Implemented in NewtonSolverIterative, NewtonSolverSparse, and NewtonSolverDense.

Here is the caller graph for this function:



10.5.4 Member Data Documentation

10.5.4.1 model

```
Model* model [protected]
```

pointer to the AMICI model object

Definition at line 53 of file newton_solver.h.

10.5.4.2 rdata

`ReturnData* rdata [protected]`

pointer to the return data object

Definition at line 55 of file newton_solver.h.

10.5.4.3 udata

`UserData* udata [protected]`

pointer to the user data object

Definition at line 57 of file newton_solver.h.

10.5.4.4 tdata

`TempData* tdata [protected]`

pointer to the temporary data object

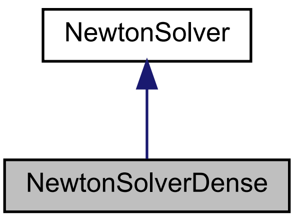
Definition at line 59 of file newton_solver.h.

10.6 NewtonSolverDense Class Reference

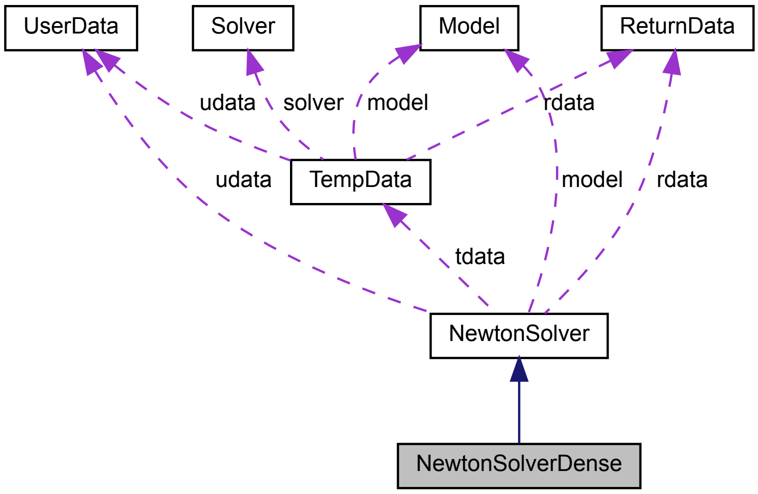
The NewtonSolverDense provides access to the dense linear solver for the Newton method.

`#include <newton_solver.h>`

Inheritance diagram for NewtonSolverDense:



Collaboration diagram for NewtonSolverDense:



Public Member Functions

- NewtonSolverDense (Model ∗model, ReturnData ∗rdata, UserData ∗udata, TempData ∗tdata)
- int solveLinearSystem (N_Vector rhs)
- int prepareLinearSystem (int ntry, int nnewt)

Additional Inherited Members

10.6.1 Detailed Description

Definition at line 66 of file newton_solver.h.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 NewtonSolverDense()

```
NewtonSolverDense (
            Model * model,
            ReturnData * rdata,
            UserData * udata,
            TempData * tdata )
```

default constructor, initializes all members with the provided objects and initializes temporary storage objects

Parameters

in	*model*	pointer to the AMICI model object **Type**: Model
in	*rdata*	pointer to the return data object **Type**: ReturnData
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData

Definition at line 173 of file newton_solver.cpp.

10.6.3 Member Function Documentation

10.6.3.1 solveLinearSystem()

```
int solveLinearSystem (
            N_Vector rhs )  [virtual]
```

Solves the linear system for the Newton step

Parameters

in,out	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Solves the linear system for the Newton step

Parameters

in,out	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

Definition at line 216 of file newton_solver.cpp.

10.6.3.2 prepareLinearSystem()

```
int prepareLinearSystem (
            int ntry,
            int nnewt )  [virtual]
```

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step

Returns

stats integer flag indicating success of the method

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step

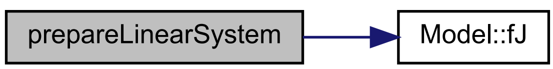
Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

Definition at line 193 of file newton_solver.cpp.

Here is the call graph for this function:

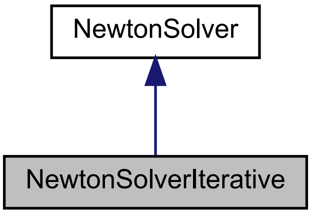


10.7 NewtonSolverIterative Class Reference

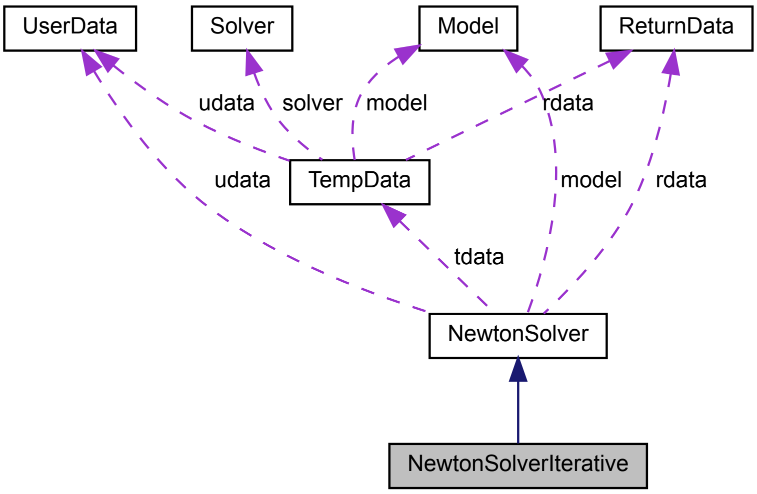
The NewtonSolverIterative provides access to the iterative linear solver for the Newton method.

```
#include <newton_solver.h>
```

Inheritance diagram for NewtonSolverIterative:



Collaboration diagram for NewtonSolverIterative:



Public Member Functions

- NewtonSolverIterative (Model *model, ReturnData *rdata, UserData *udata, TempData *tdata)
- int solveLinearSystem (N_Vector rhs)
- int prepareLinearSystem (int ntry, int nnewt)

Additional Inherited Members

10.7.1 Detailed Description

Definition at line 118 of file newton_solver.h.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 NewtonSolverIterative()

```
NewtonSolverIterative (
            Model * model,
            ReturnData * rdata,
            UserData * udata,
            TempData * tdata )
```

default constructor, initializes all members with the provided objects

Parameters

in	*model*	pointer to the AMICI model object **Type**: Model
in	*rdata*	pointer to the return data object **Type**: ReturnData
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData

Definition at line 343 of file newton_solver.cpp.

10.7.3 Member Function Documentation

10.7.3.1 solveLinearSystem()

```
int solveLinearSystem (
            N_Vector rhs )  [virtual]
```

Solves the linear system for the Newton step

Parameters

in,out	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Solves the linear system for the Newton step by passing it to linsolveSPBCG

Parameters

`in,out`	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

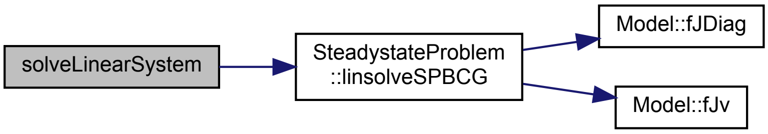
Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

Definition at line 378 of file newton_solver.cpp.

Here is the call graph for this function:



10.7.3.2 prepareLinearSystem()

```
int prepareLinearSystem (
              int ntry,
              int nnewt )   [virtual]
```

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

`in`	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
`in`	*nnewt*	integer number of current Newton step

Returns

stats integer flag indicating success of the method

Writes the Jacobian for the Newton iteration and passes it to the linear solver. Also wraps around getSensis for iterative linear solver.

Parameters

`in`	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
`in`	*nnewt*	integer number of current Newton step

Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

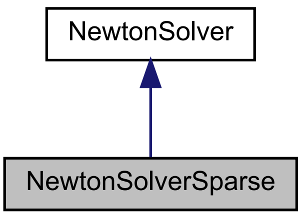
Definition at line 357 of file newton_solver.cpp.

10.8 NewtonSolverSparse Class Reference

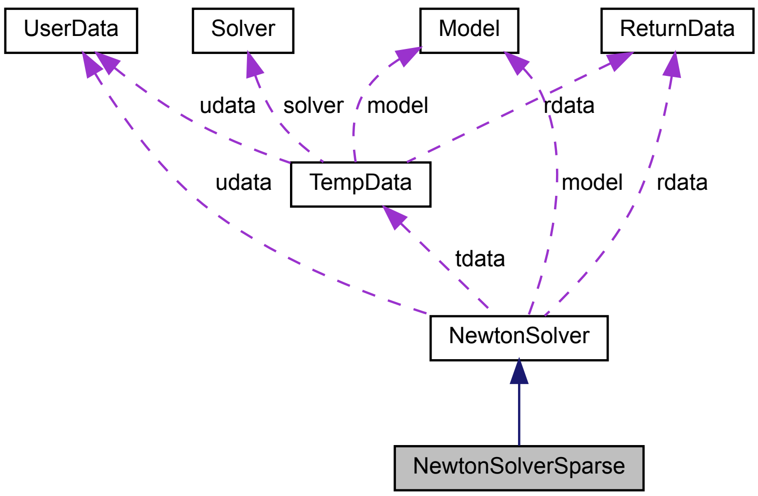
The NewtonSolverSparse provides access to the sparse linear solver for the Newton method.

`#include <newton_solver.h>`

Inheritance diagram for NewtonSolverSparse:



Collaboration diagram for NewtonSolverSparse:



Public Member Functions

- NewtonSolverSparse (Model *model, ReturnData *rdata, UserData *udata, TempData *tdata)
- int solveLinearSystem (N_Vector rhs)
- int prepareLinearSystem (int ntry, int nnewt)

Additional Inherited Members

10.8.1 Detailed Description

Definition at line 89 of file newton_solver.h.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 NewtonSolverSparse()

```
NewtonSolverSparse (
            Model * model,
            ReturnData * rdata,
            UserData * udata,
            TempData * tdata )
```

default constructor, initializes all members with the provided objects, initializes temporary storage objects and the klu solver

Parameters

in	*model*	pointer to the AMICI model object **Type**: Model
in	*rdata*	pointer to the return data object **Type**: ReturnData
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData

Definition at line 246 of file newton_solver.cpp.

10.8.3 Member Function Documentation

10.8.3.1 solveLinearSystem()

```
int solveLinearSystem (
            N_Vector rhs )  [virtual]
```

Solves the linear system for the Newton step

Parameters

in,out	*rhs*	containing the RHS of the linear system, will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Solves the linear system for the Newton step

Parameters

in	*rhs*	containing the RHS of the linear system,will be overwritten by solution to the linear system **Type**: N_Vector

Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

Definition at line 307 of file newton_solver.cpp.

10.8.3.2 prepareLinearSystem()

```
int prepareLinearSystem (
            int ntry,
            int nnewt )  [virtual]
```

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step

Returns

stats integer flag indicating success of the method

Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step

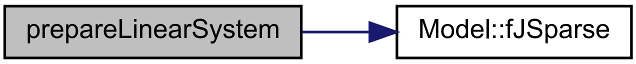
Returns

stats integer flag indicating success of the method

Implements NewtonSolver.

Definition at line 268 of file newton_solver.cpp.

Here is the call graph for this function:

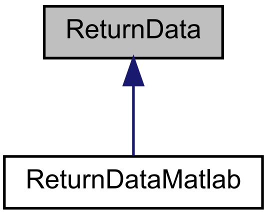


10.9 ReturnData Class Reference

struct that stores all data which is later returned by the mex function

```
#include <rdata.h>
```

Inheritance diagram for ReturnData:



Public Member Functions

- ReturnData ()
  default constructor
- ReturnData (const UserData ∗udata, const Model ∗model)
- virtual void setDefaults ()
- void invalidate ()
- void setLikelihoodSensitivityFirstOrderNaN ()
- void setLikelihoodSensitivitySecondOrderNaN ()
- int applyChainRuleFactorToSimulationResults (const UserData ∗udata, const realtype ∗unscaledParameters)
- virtual ∼ReturnData ()

Public Attributes

- double ∗ ts
- double ∗ xdot
- double ∗ J
- double ∗ z
- double ∗ sigmaz
- double ∗ sz
- double ∗ ssigmaz
- double ∗ rz
- double ∗ srz
- double ∗ s2rz
- double ∗ x
- double ∗ sx
- double ∗ y
- double ∗ sigmay
- double ∗ sy
- double ∗ ssigmay
- double ∗ numsteps
- double ∗ numstepsB
- double ∗ numrhsevals
- double ∗ numrhsevalsB
- double ∗ numerrtestfails
- double ∗ numerrtestfailsB
- double ∗ numnonlinsolvconvfails
- double ∗ numnonlinsolvconvfailsB
- double ∗ order
- double ∗ newton_status
- double ∗ newton_time
- double ∗ newton_numsteps
- double ∗ newton_numlinsteps
- double ∗ xss
- double ∗ llh
- double ∗ chi2
- double ∗ sllh
- double ∗ s2llh
- double ∗ status
- const int np
- const int nk
- const int nx
- const int nxtrue
- const int ny
- const int nytrue
- const int nz
- const int nztrue
- const int ne
- const int nJ
- const int nplist
- const int nmaxevent
- const int nt
- const int newton_maxsteps
- const AMICI_parameter_scaling pscale
- const AMICI_o2mode o2mode
- const AMICI_sensi_order sensi
- const AMICI_sensi_meth sensi_meth

Protected Member Functions

- virtual void copyFromUserData (const UserData *udata)
- virtual void initFields ()
- virtual void initField1 (double **fieldPointer, const char *fieldName, int dim)
- virtual void initField2 (double **fieldPointer, const char *fieldName, int dim1, int dim2)
- virtual void initField3 (double **fieldPointer, const char *fieldName, int dim1, int dim2, int dim3)
- virtual void initField4 (double **fieldPointer, const char *fieldName, int dim1, int dim2, int dim3, int dim4)

Protected Attributes

- bool freeFieldsOnDestruction

10.9.1 Detailed Description

NOTE: MATLAB stores multidimensional arrays in column-major order (FORTRAN-style)

Definition at line 13 of file rdata.h.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 ReturnData()

```
ReturnData (
            const UserData * udata,
            const Model * model )
```

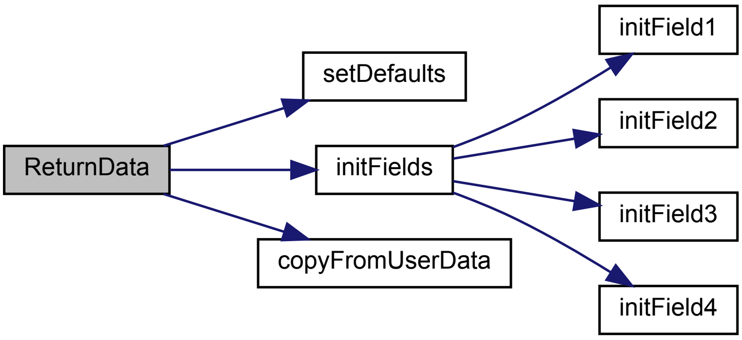
constructor that uses information from model and userdata to appropriately initialize fields

Parameters

in	*udata*	pointer to the user data struct **Type**: UserData
in	*model*	pointer to model specification object **Type**: Model

Definition at line 19 of file rdata.cpp.

Here is the call graph for this function:



10.9.2.2 ∼ReturnData()

∼ReturnData ( ) [virtual]

default destructor

Definition at line 258 of file rdata.cpp.

10.9.3 Member Function Documentation

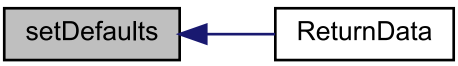
10.9.3.1 setDefaults()

void setDefaults ( ) [virtual]

initialize all member fields will nullpointers

Definition at line 40 of file rdata.cpp.

Here is the caller graph for this function:



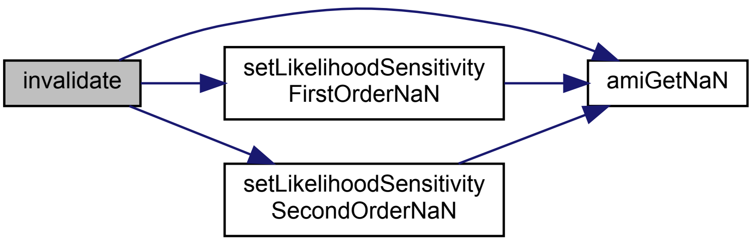
10.9.3.2 invalidate()

```
void invalidate ( )
```

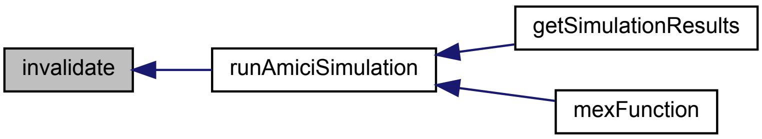
routine to set likelihood and respective sensitivities to NaN (typically after integration failure)

Definition at line 57 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



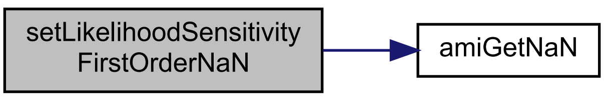
10.9.3.3 setLikelihoodSensitivityFirstOrderNaN()

```
void setLikelihoodSensitivityFirstOrderNaN ( )
```

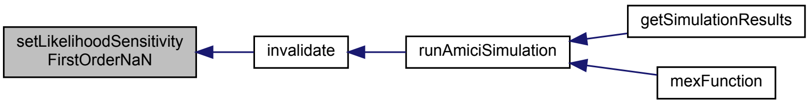
routine to set first order sensitivities to NaN (typically after integration failure)

Definition at line 71 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



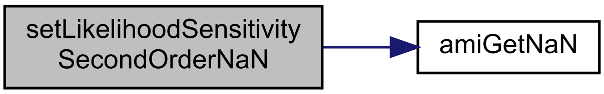
10.9.3.4 setLikelihoodSensitivitySecondOrderNaN()

```
void setLikelihoodSensitivitySecondOrderNaN ( )
```

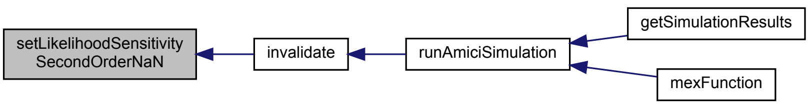
routine to set second order sensitivities to NaN (typically after integration failure)

Definition at line 78 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.9.3.5 applyChainRuleFactorToSimulationResults()

```
int applyChainRuleFactorToSimulationResults (
            const UserData * udata,
            const realtype * unscaledParameters )
```

applies the chain rule to account for parameter transformation in the sensitivities of simulation results

Parameters

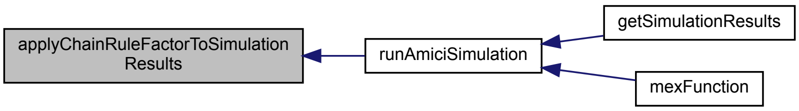
in	*udata*	pointer to the user data struct **Type**: UserData
in	*unscaledParameters*	pointer to the non-transformed parameters **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Definition at line 85 of file rdata.cpp.

Here is the caller graph for this function:



10.9.3.6 copyFromUserData()

```
void copyFromUserData (
            const UserData * udata )  [protected], [virtual]
```

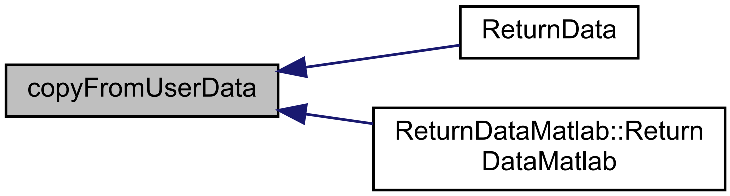
copies measurement timepoints from UserData object

Parameters

in	*udata*	pointer to the user data struct **Type**: UserData

Definition at line 337 of file rdata.cpp.

Here is the caller graph for this function:



10.9.3.7 initFields()

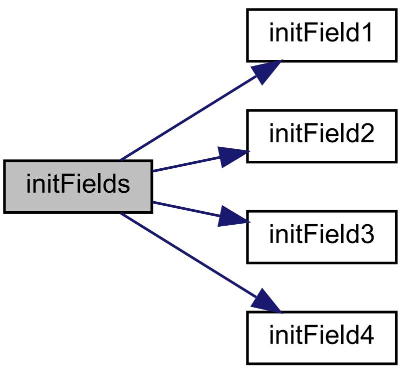
```
void initFields ( ) [protected], [virtual]
```

initialises sol object with the corresponding fields

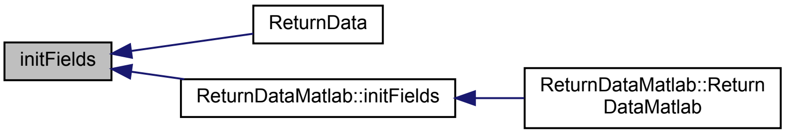
Reimplemented in ReturnDataMatlab.

Definition at line 345 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.9.3.8 initField1()

```
void initField1 (
            double ** fieldPointer,
            const char * fieldName,
            int dim )  [protected], [virtual]
```

initialise vector and attach to the field

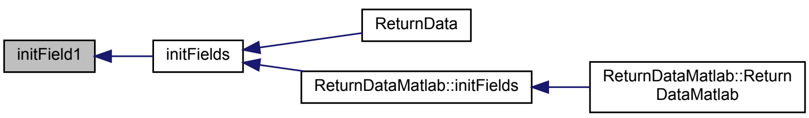
Parameters

fieldPointer	pointer of the field to which the vector will be attached
fieldName	Name of the field to which the vector will be attached
dim	number of elements in the vector

Reimplemented in ReturnDataMatlab.

Definition at line 420 of file rdata.cpp.

Here is the caller graph for this function:



10.9.3.9 initField2()

```
void initField2 (
            double ** fieldPointer,
            const char * fieldName,
            int dim1,
            int dim2 )  [protected], [virtual]
```

initialise matrix and attach to the field

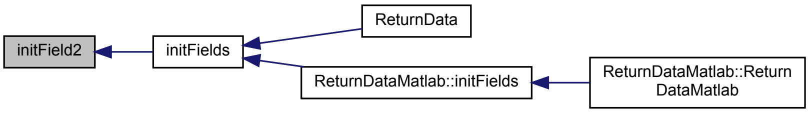
Parameters

fieldPointer	pointer of the field to which the matrix will be attached
fieldName	Name of the field to which the matrix will be attached
dim1	number of rows in the matrix
dim2	number of columns in the matrix

Reimplemented in ReturnDataMatlab.

Definition at line 431 of file rdata.cpp.

Here is the caller graph for this function:



10.9.3.10 initField3()

```
void initField3 (
            double ** fieldPointer,
            const char * fieldName,
            int dim1,
            int dim2,
            int dim3 )  [protected], [virtual]
```

initialise 3D tensor and attach to the field

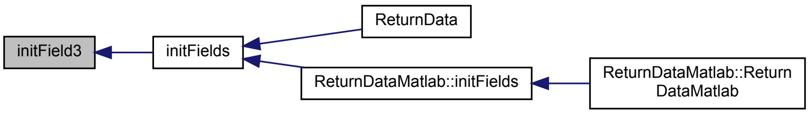
Parameters

fieldPointer	pointer of the field to which the tensor will be attached
fieldName	Name of the field to which the tensor will be attached
dim1	number of rows in the tensor
dim2	number of columns in the tensor
dim3	number of elements in the third dimension of the tensor

Reimplemented in ReturnDataMatlab.

Definition at line 443 of file rdata.cpp.

Here is the caller graph for this function:



10.9.3.11 initField4()

```
void initField4 (
            double ** fieldPointer,
            const char * fieldName,
            int dim1,
            int dim2,
            int dim3,
            int dim4 )  [protected], [virtual]
```

initialise 4D tensor and attach to the field

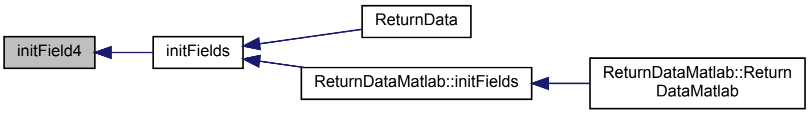
Parameters

fieldPointer	pointer of the field to which the tensor will be attached
fieldName	Name of the field to which the tensor will be attached
dim1	number of rows in the tensor
dim2	number of columns in the tensor
dim3	number of elements in the third dimension of the tensor
dim4	number of elements in the fourth dimension of the tensor

Reimplemented in ReturnDataMatlab.

Definition at line 457 of file rdata.cpp.

Here is the caller graph for this function:



10.9.4 Member Data Documentation

10.9.4.1 ts

```
double* ts
```

timepoints (dimension: nt)

Definition at line 33 of file rdata.h.

10.9.4.2 xdot

```
double* xdot
```

time derivative (dimension: nx)

Definition at line 36 of file rdata.h.

10.9.4.3 J

```
double* J
```

Jacobian of differential equation right hand side (dimension: nx x nx, column-major)

Definition at line 40 of file rdata.h.

10.9.4.4 z

```
double* z
```

event output (dimension: nmaxevent x nz, column-major)

Definition at line 43 of file rdata.h.

10.9.4.5 sigmaz

```
double* sigmaz
```

event output sigma standard deviation (dimension: nmaxevent x nz, column-major)

Definition at line 47 of file rdata.h.

10.9.4.6 sz

```
double* sz
```

parameter derivative of event output (dimension: nmaxevent x nz, column-major)

Definition at line 51 of file rdata.h.

10.9.4.7 ssigmaz

```
double* ssigmaz
```

parameter derivative of event output standard deviation (dimension: nmaxevent x nz, column-major)

Definition at line 55 of file rdata.h.

10.9.4.8 rz

```
double* rz
```

event trigger output (dimension: nmaxevent x nz, column-major)

Definition at line 58 of file rdata.h.

10.9.4.9 srz

```
double* srz
```

parameter derivative of event trigger output (dimension: nmaxevent x nz x nplist, column-major)

Definition at line 62 of file rdata.h.

10.9.4.10 s2rz

```
double* s2rz
```

second order parameter derivative of event trigger output (dimension: nmaxevent x nztrue x nplist x nplist, column-major)

Definition at line 66 of file rdata.h.

10.9.4.11 x

```
double* x
```

state (dimension: nt x nx, column-major)

Definition at line 69 of file rdata.h.

10.9.4.12 sx

```
double* sx
```

parameter derivative of state (dimension: nt x nx x nplist, column-major)

Definition at line 73 of file rdata.h.

10.9.4.13 y

```
double* y
```

observable (dimension: nt x ny, column-major)

Definition at line 76 of file rdata.h.

10.9.4.14 sigmay

```
double* sigmay
```

observable standard deviation (dimension: nt x ny, column-major)

Definition at line 79 of file rdata.h.

10.9.4.15 sy

```
double* sy
```

parameter derivative of observable (dimension: nt x ny x nplist, column-major)

Definition at line 83 of file rdata.h.

10.9.4.16 ssigmay

```
double* ssigmay
```

parameter derivative of observable standard deviation (dimension: nt x ny x nplist, column-major)

Definition at line 87 of file rdata.h.

10.9.4.17 numsteps

```
double* numsteps
```

number of integration steps forward problem (dimension: nt)

Definition at line 90 of file rdata.h.

10.9.4.18 numstepsB

```
double* numstepsB
```

number of integration steps backward problem (dimension: nt)

Definition at line 93 of file rdata.h.

10.9.4.19 numrhsevals

```
double* numrhsevals
```

number of right hand side evaluations forward problem (dimension: nt)

Definition at line 96 of file rdata.h.

10.9.4.20 numrhsevalsB

```
double* numrhsevalsB
```

number of right hand side evaluations backwad problem (dimension: nt)

Definition at line 99 of file rdata.h.

10.9.4.21 numerrtestfails

```
double* numerrtestfails
```

number of error test failures forward problem (dimension: nt)

Definition at line 102 of file rdata.h.

10.9.4.22 numerrtestfailsB

```
double* numerrtestfailsB
```

number of error test failures backwad problem (dimension: nt)

Definition at line 105 of file rdata.h.

10.9.4.23 numnonlinsolvconvfails

```
double* numnonlinsolvconvfails
```

number of linear solver convergence failures forward problem (dimension: nt)

Definition at line 109 of file rdata.h.

10.9.4.24 numnonlinsolvconvfailsB

```
double* numnonlinsolvconvfailsB
```

number of linear solver convergence failures backwad problem (dimension: nt)

Definition at line 113 of file rdata.h.

10.9.4.25 order

```
double* order
```

employed order forward problem (dimension: nt)

Definition at line 116 of file rdata.h.

10.9.4.26 newton_status

```
double* newton_status
```

flag indicating success of Newton solver

Definition at line 119 of file rdata.h.

10.9.4.27 newton_time

```
double* newton_time
```

computation time of the Newton solver [s]

Definition at line 122 of file rdata.h.

10.9.4.28 newton_numsteps

```
double* newton_numsteps
```

number of Newton steps for steady state problem

Definition at line 125 of file rdata.h.

10.9.4.29 newton_numlinsteps

```
double* newton_numlinsteps
```

number of linear steps by Newton step for steady state problem

Definition at line 128 of file rdata.h.

10.9.4.30 xss

```
double* xss
```

steady state found be Newton solver

Definition at line 131 of file rdata.h.

10.9.4.31 llh

```
double* llh
```

likelihood value (double[1])

Definition at line 134 of file rdata.h.

10.9.4.32 chi2

```
double* chi2
```

chi2 value (double[1])

Definition at line 137 of file rdata.h.

10.9.4.33 sllh

```
double* sllh
```

parameter derivative of likelihood (dimension: nplist)

Definition at line 140 of file rdata.h.

10.9.4.34 s2llh

```
double* s2llh
```

second order parameter derivative of likelihood (dimension: (nJ-1) x nplist, column-major)

Definition at line 144 of file rdata.h.

10.9.4.35 status

```
double* status
```

status code (double[1])

Definition at line 147 of file rdata.h.

10.9.4.36 freeFieldsOnDestruction

```
bool freeFieldsOnDestruction [protected]
```

flag indicating whether memory for fields needs to be freed on destruction

Definition at line 167 of file rdata.h.

10.9.4.37 np

```
const int np
```

total number of model parameters

Definition at line 171 of file rdata.h.

10.9.4.38 nk

```
const int nk
```

number of fixed parameters

Definition at line 173 of file rdata.h.

10.9.4.39 nx

```
const int nx
```

number of states

Definition at line 175 of file rdata.h.

10.9.4.40 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 177 of file rdata.h.

10.9.4.41 ny

```
const int ny
```

number of observables

Definition at line 179 of file rdata.h.

10.9.4.42 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 181 of file rdata.h.

10.9.4.43 nz

```
const int nz
```

number of event outputs

Definition at line 183 of file rdata.h.

10.9.4.44 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 185 of file rdata.h.

10.9.4.45 ne

```
const int ne
```

number of events

Definition at line 187 of file rdata.h.

10.9.4.46 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 189 of file rdata.h.

10.9.4.47 nplist

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 192 of file rdata.h.

10.9.4.48 nmaxevent

```
const int nmaxevent
```

maximal number of occuring events (for every event type)

Definition at line 194 of file rdata.h.

10.9.4.49 nt

```
const int nt
```

number of considered timepoints

Definition at line 196 of file rdata.h.

10.9.4.50 newton_maxsteps

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 198 of file rdata.h.

10.9.4.51 pscale

```
const AMICI_parameter_scaling pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 200 of file rdata.h.

10.9.4.52 o2mode

```
const AMICI_o2mode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 202 of file rdata.h.

10.9.4.53 sensi

```
const AMICI_sensi_order sensi
```

sensitivity order

Definition at line 204 of file rdata.h.

10.9.4.54 sensi_meth

```
const AMICI_sensi_meth sensi_meth
```

sensitivity method

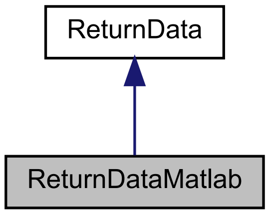
Definition at line 206 of file rdata.h.

10.10 ReturnDataMatlab Class Reference

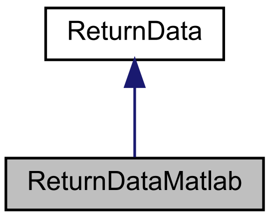
The ReturnDataMatlab class sets up ReturnData to be returned by the MATLAB mex functions. Memory is allocated using matlab functions.

```
#include <returndata_matlab.h>
```

Inheritance diagram for ReturnDataMatlab:



Collaboration diagram for ReturnDataMatlab:



Public Member Functions

- ReturnDataMatlab (const UserData *udata, const Model *model)

Public Attributes

- mxArray * mxsol

Protected Member Functions

- void initFields ()
- virtual void initField1 (double **fieldPointer, const char *fieldName, int dim)
- virtual void initField2 (double **fieldPointer, const char *fieldName, int dim1, int dim2)
- virtual void initField3 (double **fieldPointer, const char *fieldName, int dim1, int dim2, int dim3)
- virtual void initField4 (double **fieldPointer, const char *fieldName, int dim1, int dim2, int dim3, int dim4)

Additional Inherited Members

10.10.1 Detailed Description

Definition at line 15 of file returndata_matlab.h.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 ReturnDataMatlab()

```
ReturnDataMatlab (
            const UserData * udata,
            const Model * model )
```

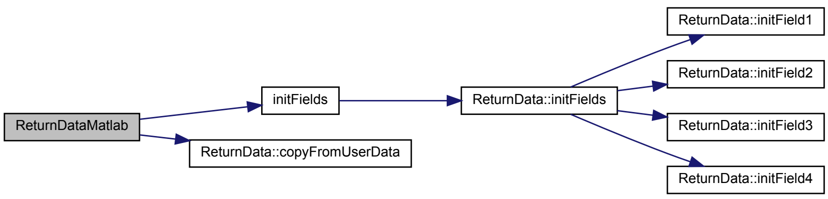
initialises the returnData struct, initialises the fields and copies model dimensions from the udata struct

Parameters

in	*udata*	pointer to the user data struct **Type**: UserData
in	*model*	pointer to model specification object **Type**: Model

Definition at line 3 of file returndata_matlab.cpp.

Here is the call graph for this function:



10.10.3 Member Function Documentation

10.10.3.1 initFields()

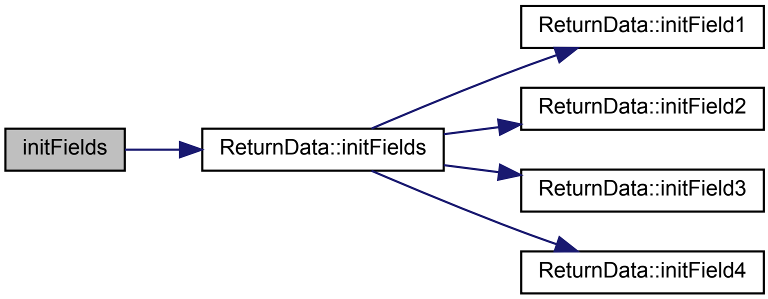
```
void initFields ( ) [protected], [virtual]
```

initialises sol object with the corresponding fields

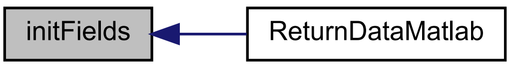
Reimplemented from ReturnData.

Definition at line 17 of file returndata_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.10.3.2 initField1()

```
void initField1 (
            double ** fieldPointer,
            const char * fieldName,
            int dim )  [protected], [virtual]
```

initialise vector and attach to the field

Parameters

fieldPointer	pointer of the field to which the vector will be attached
fieldName	Name of the field to which the vector will be attached
dim	number of elements in the vector

Reimplemented from ReturnData.

Definition at line 63 of file returndata_matlab.cpp.

10.10.3.3 initField2()

```
void initField2 (
            double ** fieldPointer,
            const char * fieldName,
            int dim1,
            int dim2 )  [protected], [virtual]
```

initialise matrix and attach to the field

Parameters

fieldPointer	pointer of the field to which the matrix will be attached
fieldName	Name of the field to which the matrix will be attached
dim1	number of rows in the matrix
dim2	number of columns in the matrix

Reimplemented from ReturnData.

Definition at line 80 of file returndata_matlab.cpp.

10.10.3.4 initField3()

```
void initField3 (
            double ** fieldPointer,
            const char * fieldName,
            int dim1,
            int dim2,
            int dim3 )  [protected], [virtual]
```

initialise 3D tensor and attach to the field

Parameters

fieldPointer	pointer of the field to which the tensor will be attached
fieldName	Name of the field to which the tensor will be attached
dim1	number of rows in the tensor
dim2	number of columns in the tensor
dim3	number of elements in the third dimension of the tensor

Reimplemented from ReturnData.

Definition at line 98 of file returndata_matlab.cpp.

10.10.3.5 initField4()

```
void initField4 (
            double ** fieldPointer,
```

```
const char * fieldName,
int dim1,
int dim2,
int dim3,
int dim4 )  [protected], [virtual]
```

initialise 4D tensor and attach to the field

Parameters

fieldPointer	pointer of the field to which the tensor will be attached
fieldName	Name of the field to which the tensor will be attached
dim1	number of rows in the tensor
dim2	number of columns in the tensor
dim3	number of elements in the third dimension of the tensor
dim4	number of elements in the fourth dimension of the tensor

Reimplemented from ReturnData.

Definition at line 118 of file returndata_matlab.cpp.

10.10.4 Member Data Documentation

10.10.4.1 mxsol

```
mxArray* mxsol
```

sol struct that is passed back to matlab

Definition at line 22 of file returndata_matlab.h.

10.11 Solver Class Reference

Solver class.

```
#include <amici_solver.h>
```

Public Member Functions

- int setupAMI (UserData *udata, TempData *tdata, Model *model)

  setupAMIs initialises the ami memory object
- int setupAMIB (UserData *udata, TempData *tdata, Model *model)
- virtual int AMIGetSens (realtype *tret, N_Vector *yySout)=0
- int getDiagnosis (const int it, ReturnData *rdata)
- int getDiagnosisB (const int it, ReturnData *rdata, const TempData *tdata)
- virtual int AMIGetRootInfo (int *rootsfound)=0
- virtual int AMIReInit (realtype t0, N_Vector yy0, N_Vector yp0)=0
- virtual int AMISensReInit (int ism, N_Vector *yS0, N_Vector *ypS0)=0

- virtual int AMICalcIC (realtype tout1)=0
- virtual int AMICalcICB (int which, realtype tout1, N_Vector xB, N_Vector dxB)=0
- virtual int AMISolve (realtype tout, N_Vector yret, N_Vector ypret, realtype ∗tret, int itask)=0
- virtual int AMISolveF (realtype tout, N_Vector yret, N_Vector ypret, realtype ∗tret, int itask, int ∗ncheckPtr)=0
- virtual int AMISolveB (realtype tBout, int itaskB)=0
- virtual int AMISetStopTime (realtype tstop)=0
- virtual int AMIReInitB (int which, realtype tB0, N_Vector yyB0, N_Vector ypB0)=0
- virtual int AMIGetB (int which, realtype ∗tret, N_Vector yy, N_Vector yp)=0
- virtual int AMIGetQuadB (int which, realtype ∗tret, N_Vector qB)=0
- virtual int AMIQuadReInitB (int which, N_Vector yQB0)=0
- virtual int turnOffRootFinding ()=0

Protected Member Functions

- virtual int wrap_init (N_Vector x, N_Vector dx, realtype t)=0
- virtual int wrap_binit (int which, N_Vector xB, N_Vector dxB, realtype t)=0
- virtual int wrap_qbinit (int which, N_Vector qBdot)=0
- virtual int wrap_RootInit (int ne)=0
- virtual int wrap_SensInit1 (N_Vector ∗sx, N_Vector ∗sdx, const UserData ∗udata)=0
- virtual int wrap_SetDenseJacFn ()=0
- virtual int wrap_SetSparseJacFn ()=0
- virtual int wrap_SetBandJacFn ()=0
- virtual int wrap_SetJacTimesVecFn ()=0
- virtual int wrap_SetDenseJacFnB (int which)=0
- virtual int wrap_SetSparseJacFnB (int which)=0
- virtual int wrap_SetBandJacFnB (int which)=0
- virtual int wrap_SetJacTimesVecFnB (int which)=0
- virtual void ∗ AMICreate (int lmm, int iter)=0
- virtual int AMISStolerances (double rtol, double atol)=0
- virtual int AMISensEEtolerances ()=0
- virtual int AMISetSensErrCon (bool error_corr)=0
- virtual int AMISetQuadErrConB (int which, bool flag)=0
- virtual int AMISetErrHandlerFn ()=0
- virtual int AMISetUserData (void ∗user_data)=0
- virtual int AMISetUserDataB (int which, void ∗user_data)=0
- virtual int AMISetMaxNumSteps (long int mxsteps)=0
- virtual int AMISetMaxNumStepsB (int which, long int mxstepsB)=0
- virtual int AMISetStabLimDet (int stldet)=0
- virtual int AMISetStabLimDetB (int which, int stldet)=0
- virtual int AMISetId (Model ∗model)=0
- virtual int AMISetSuppressAlg (bool flag)=0
- virtual int AMISetSensParams (realtype ∗p, realtype ∗pbar, int ∗plist)=0
- virtual int AMIGetDky (realtype t, int k, N_Vector dky)=0
- virtual void AMIFree ()=0
- virtual int AMIAdjInit (long int steps, int interp)=0
- virtual int AMICreateB (int lmm, int iter, int ∗which)=0
- virtual int AMISStolerancesB (int which, realtype relTolB, realtype absTolB)=0
- virtual int AMIQuadSStolerancesB (int which, realtype reltolQB, realtype abstolQB)=0
- virtual int AMIDense (int nx)=0
- virtual int AMIDenseB (int which, int nx)=0
- virtual int AMIBand (int nx, int ubw, int lbw)=0
- virtual int AMIBandB (int which, int nx, int ubw, int lbw)=0
- virtual int AMIDiag ()=0
- virtual int AMIDiagB (int which)=0

- virtual int AMISpgmr (int prectype, int maxl)=0
- virtual int AMISpgmrB (int which, int prectype, int maxl)=0
- virtual int AMISpbcg (int prectype, int maxl)=0
- virtual int AMISpbcgB (int which, int prectype, int maxl)=0
- virtual int AMISptfqmr (int prectype, int maxl)=0
- virtual int AMISptfqmrB (int which, int prectype, int maxl)=0
- virtual int AMIKLU (int nx, int nnz, int sparsetype)=0
- virtual int AMIKLUSetOrdering (int ordering)=0
- virtual int AMIKLUSetOrderingB (int which, int ordering)=0
- virtual int AMIKLUB (int which, int nx, int nnz, int sparsetype)=0
- virtual int AMIGetNumSteps (void *ami_mem, long int *numsteps)=0
- virtual int AMIGetNumRhsEvals (void *ami_mem, long int *numrhsevals)=0
- virtual int AMIGetNumErrTestFails (void *ami_mem, long int *numerrtestfails)=0
- virtual int AMIGetNumNonlinSolvConvFails (void *ami_mem, long int *numnonlinsolvconvfails)=0
- virtual int AMIGetLastOrder (void *ami_mem, int *order)=0
- virtual void * AMIGetAdjBmem (void *ami_mem, int which)=0
- int setLinearSolver (const UserData *udata, Model *model)

Static Protected Member Functions

- static void wrap_ErrHandlerFn (int error_code, const char *module, const char *function, char *msg, void *eh_data)

Protected Attributes

- void * ami_mem = nullptr

10.11.1 Detailed Description

provides a generic interface to CVode and IDA solvers, individual realizations are realized in the CVodeSolver and the IDASolver class.

Definition at line 16 of file amici_solver.h.

10.11.2 Member Function Documentation

10.11.2.1 setupAMI()

```
int setupAMI (
            UserData * udata,
            TempData * tdata,
            Model * model )
```

Parameters

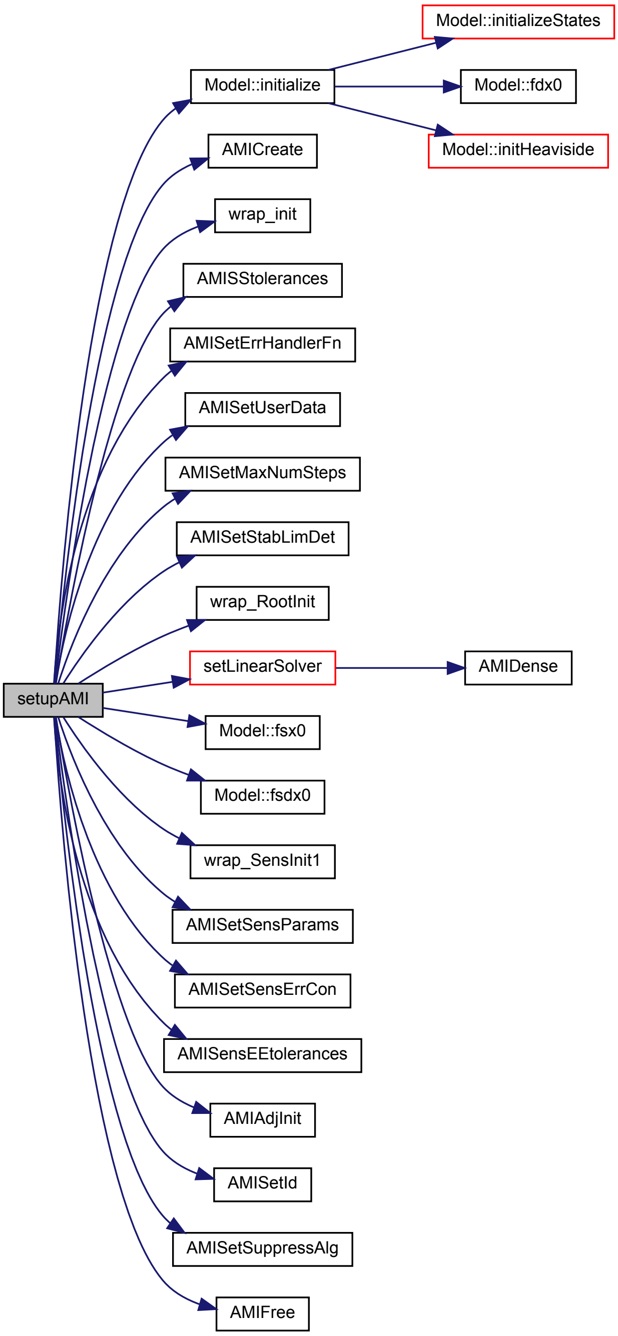
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData
in	*model*	pointer to the model object **Type**: Model

Returns

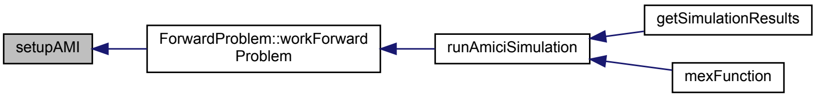
status flag indicating successful execution

Definition at line 22 of file amici_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.2 setupAMIB()

```
int setupAMIB (
            UserData * udata,
            TempData * tdata,
            Model * model )
```

setupAMIB initialises the AMI memory object for the backwards problem

Parameters

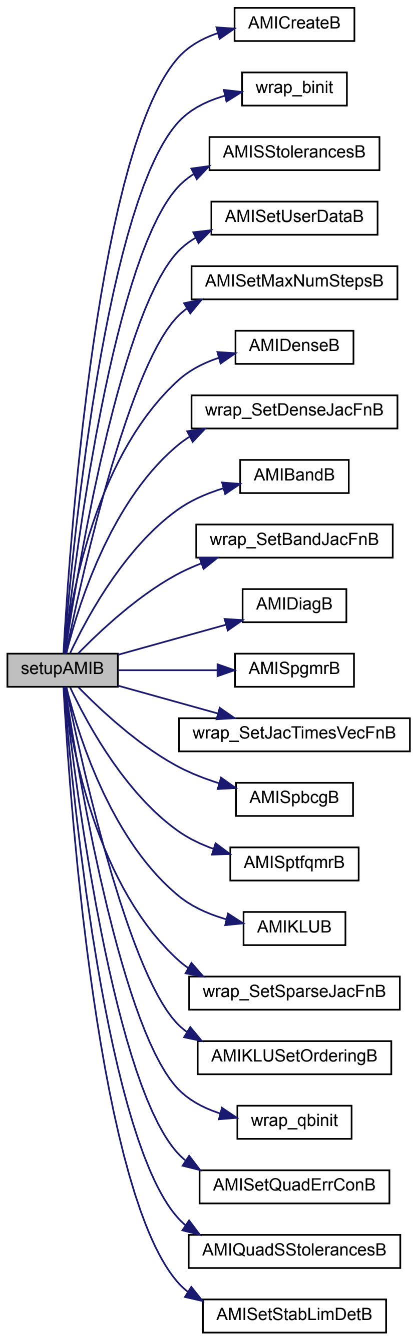
in	*udata*	pointer to the user data object **Type**: UserData
in	*tdata*	pointer to the temporary data object **Type**: TempData
in	*model*	pointer to the model object **Type**: Model

Returns

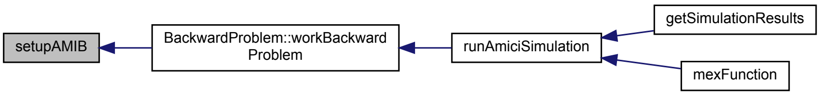
status flag indicating successful execution

Definition at line 153 of file amici_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.3 AMIGetSens()

```
virtual int AMIGetSens (
            realtype * tret,
            N_Vector * yySout ) [pure virtual]
```

AMIGetSens extracts diagnosis information from solver memory block and writes them into the return data object

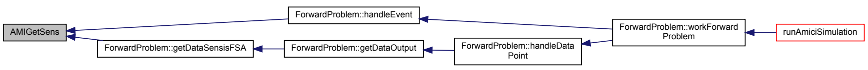
Parameters

in	*tret*	time at which the sensitivities should be computed
out	*yySout*	vector with sensitivities **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.4 getDiagnosis()

```
int getDiagnosis (
            const int it,
            ReturnData * rdata )
```

getDiagnosis extracts diagnosis information from solver memory block and writes them into the return data object

Parameters

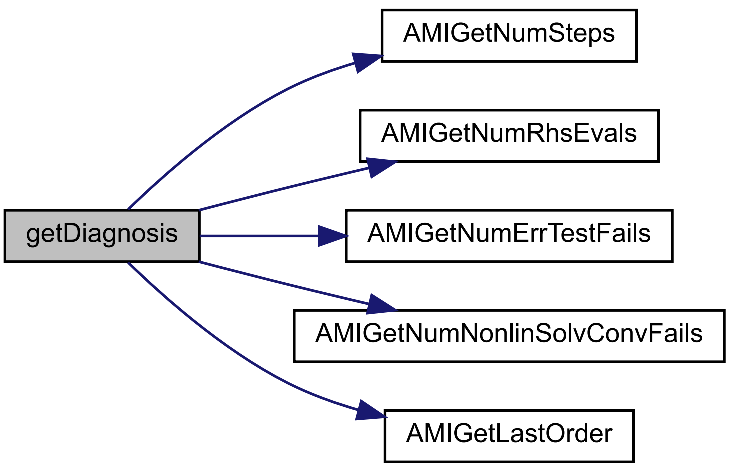
in	*it*	time-point index **Type**: int
out	*rdata*	pointer to the return data object **Type**: ReturnData

Returns

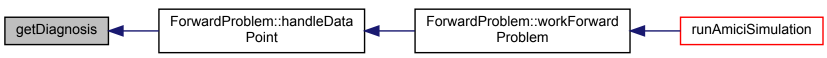
status flag indicating success of execution
Type: int

Definition at line 410 of file amici_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.5 getDiagnosisB()

```
int getDiagnosisB (
            const int it,
            ReturnData * rdata,
            const TempData * tdata )
```

getDiagnosisB extracts diagnosis information from solver memory block and writes them into the return data object for the backward problem

Parameters

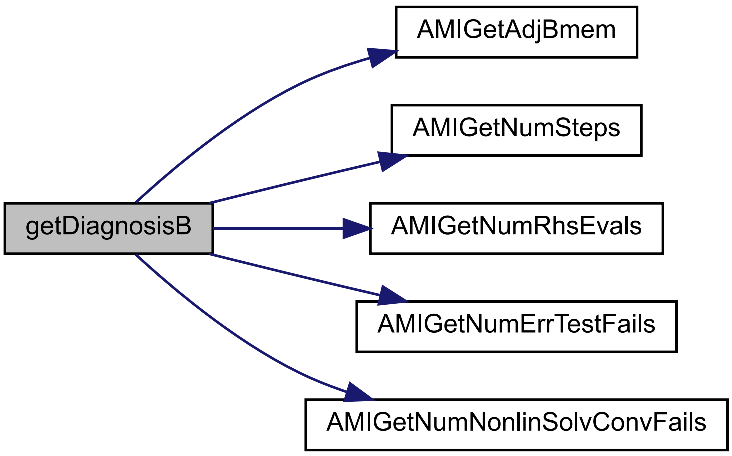
in	*it*	time-point index **Type**: int
out	*rdata*	pointer to the return data object **Type**: ReturnData
out	*tdata*	pointer to the temporary data object **Type**: TempData

Returns

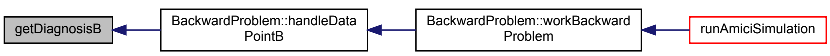
status flag indicating success of execution
Type: int

Definition at line 452 of file amici_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.2.6 AMIGetRootInfo()

```
virtual int AMIGetRootInfo (
            int * rootsfound )  [pure virtual]
```

AMIGetRootInfo extracts information which event occured

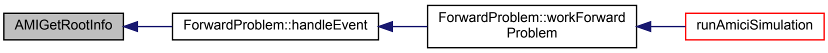
Parameters

out	*rootsfound*	array with flags indicating whether the respective event occured

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.7 AMIReInit()

```
virtual int AMIReInit (
            realtype t0,
            N_Vector yy0,
            N_Vector yp0 )  [pure virtual]
```

AMIReInit reinitializes the states in the solver after an event occurence

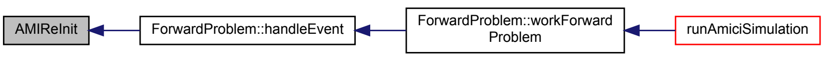
Parameters

in	*t0*	new timepoint **Type**: realtype
in	*yy0*	new state variables **Type**: N_Vector
in	*yp0*	new derivative state variables (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.8 AMISensReInit()

```
virtual int AMISensReInit (
            int ism,
            N_Vector * yS0,
            N_Vector * ypS0 ) [pure virtual]
```

AMISensReInit reinitializes the state sensitivites in the solver after an event occurence

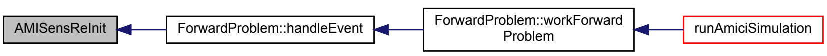
Parameters

in	*ism*	sensitivity mode **Type**: realtype
in	*yS0*	new state sensitivity **Type**: N_Vector
in	*ypS0*	new derivative state sensitivities (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.9 AMICalcIC()

```
virtual int AMICalcIC (
            realtype tout1 ) [pure virtual]
```

AMICalcIC calculates consistent initial conditions, assumes initial states to be correct (DAE only)

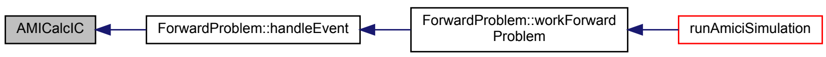
Parameters

in	*tout1*	next timepoint to be computed (sets timescale) **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.10 AMICalcICB()

```
virtual int AMICalcICB (
            int which,
            realtype tout1,
            N_Vector xB,
            N_Vector dxB )  [pure virtual]
```

AMICalcIBC calculates consistent initial conditions for the backwards problem, assumes initial states to be correct (DAE only)

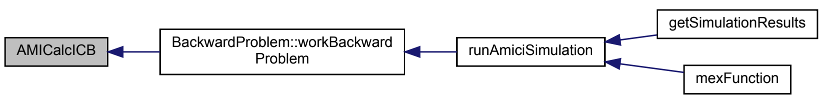
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*tout1*	next timepoint to be computed (sets timescale) **Type**: realtype
in	*xB*	states of final solution of the forward problem **Type**: N_Vector
in	*dxB*	derivative states of final solution of the forward problem (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.11 AMISolve()

```
virtual int AMISolve (
            realtype tout,
            N_Vector yret,
            N_Vector ypret,
            realtype * tret,
            int itask )  [pure virtual]
```

AMISolve solves the forward problem until a predefined timepoint

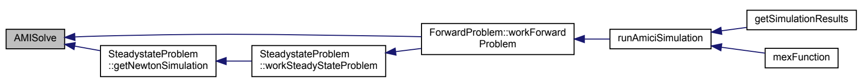
Parameters

in	*tout*	timepoint until which simulation should be performed **Type**: realtype
in	*yret*	states **Type**: N_Vector
in	*ypret*	derivative states (DAE only) **Type**: N_Vector
in,out	*tret*	pointer to the time variable **Type**: realtype
in	*itask*	task identifier, can be CV_NORMAL or CV_ONE_STEP **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.12 AMISolveF()

```
virtual int AMISolveF (
            realtype tout,
            N_Vector yret,
            N_Vector ypret,
            realtype * tret,
            int itask,
            int * ncheckPtr )  [pure virtual]
```

AMISolveF solves the forward problem until a predefined timepoint (adjoint only)

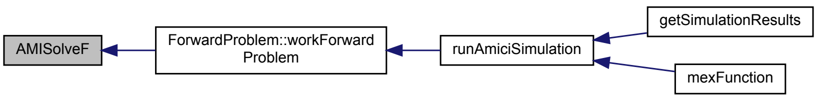
Parameters

in	*tout*	timepoint until which simulation should be performed **Type**: realtype
in	*yret*	states **Type**: N_Vector
in	*ypret*	derivative states (DAE only) **Type**: N_Vector
in,out	*tret*	pointer to the time variable **Type**: realtype
in	*itask*	task identifier, can be CV_NORMAL or CV_ONE_STEP **Type**: realtype
in	*ncheckPtr*	pointer to a number that counts the internal checkpoints **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.13 AMISolveB()

```
virtual int AMISolveB (
            realtype tBout,
            int itaskB )  [pure virtual]
```

AMISolveB solves the backward problem until a predefined timepoint (adjoint only)

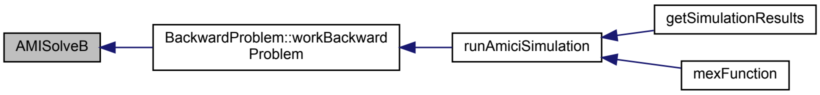
Parameters

in	*tBout*	timepoint until which simulation should be performed **Type**: realtype
in	*itaskB*	task identifier, can be CV_NORMAL or CV_ONE_STEP **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.14 AMISetStopTime()

```
virtual int AMISetStopTime (
            realtype tstop )  [pure virtual]
```

AMISetStopTime sets a timepoint at which the simulation will be stopped

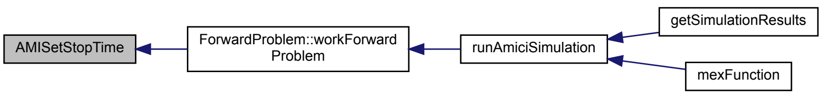
Parameters

in	*tstop*	timepoint until which simulation should be performed **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.15 AMIReInitB()

```
virtual int AMIReInitB (
            int which,
            realtype tB0,
            N_Vector yyB0,
            N_Vector ypB0 )  [pure virtual]
```

AMIReInitB reinitializes the adjoint states after an event occurence

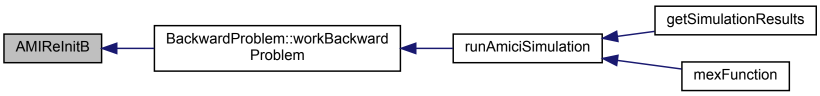
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*tB0*	new timepoint **Type**: realtype
in	*yyB0*	new adjoint state variables **Type**: N_Vector
in	*ypB0*	new adjoint derivative state variables (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.16 AMIGetB()

```
virtual int AMIGetB (
            int which,
            realtype * tret,
            N_Vector yy,
            N_Vector yp ) [pure virtual]
```

AMIGetB returns the current adjoint states

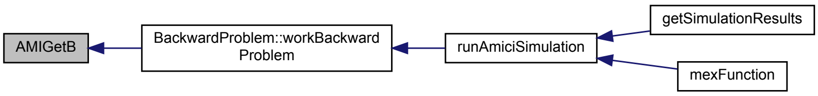
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*tret*	time at which the adjoint states should be computed
in	*yy*	adjoint state variables **Type**: N_Vector
in	*yp*	adjoint derivative state variables (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.17 AMIGetQuadB()

```
virtual int AMIGetQuadB (
            int which,
            realtype * tret,
            N_Vector qB ) [pure virtual]
```

AMIGetQuadB returns the current adjoint states

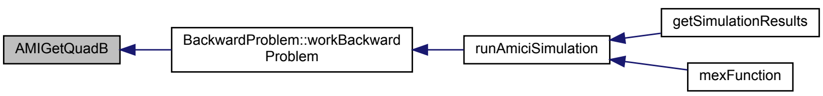
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*tret*	time at which the adjoint states should be computed
in	*qB*	adjoint quadrature state variables **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.18 AMIQuadReInitB()

```
virtual int AMIQuadReInitB (
            int which,
            N_Vector yQB0 ) [pure virtual]
```

AMIReInitB reinitializes the adjoint states after an event occurence

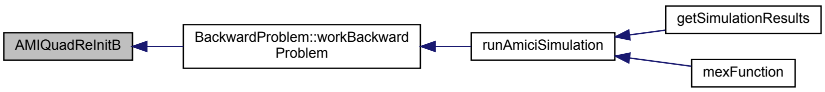
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*yQB0*	new adjoint quadrature state variables **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.19 turnOffRootFinding()

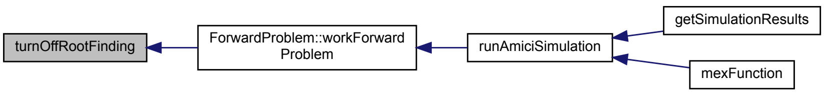
```
virtual int turnOffRootFinding ( )  [pure virtual]
```

turnOffRootFinding disables rootfinding

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.20 wrap_init()

```
virtual int wrap_init (
            N_Vector x,
            N_Vector dx,
            realtype t )  [protected], [pure virtual]
```

wrap_init initialises the states at the specified initial timepoint

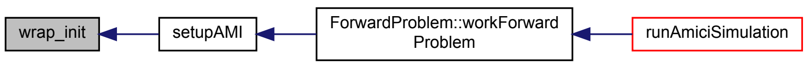
Parameters

in	*x*	initial state variables **Type**: N_Vector
in	*dx*	initial derivative state variables (DAE only) **Type**: N_Vector
in	*t*	initial timepoint **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.21 wrap_binit()

```
virtual int wrap_binit (
            int which,
            N_Vector xB,
            N_Vector dxB,
            realtype t ) [protected], [pure virtual]
```

wrap_binit initialises the adjoint states at the specified final timepoint

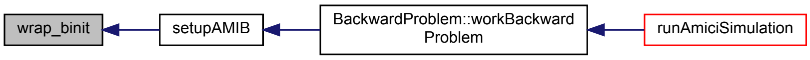
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*xB*	initial adjoint state variables **Type**: N_Vector
in	*dxB*	initial adjoint derivative state variables (DAE only) **Type**: N_Vector
in	*t*	final timepoint **Type**: realtype

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.22 wrap_qbinit()

```
virtual int wrap_qbinit (
            int which,
            N_Vector qBdot )  [protected], [pure virtual]
```

wrap_qbinit initialises the quadrature states at the specified final timepoint

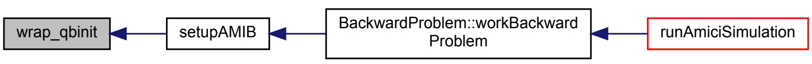
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*qBdot*	initial adjoint quadrature state variables **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.23 wrap_RootInit()

```
virtual int wrap_RootInit (
            int ne )  [protected], [pure virtual]
```

wrap_RootInit initialises the rootfinding for events

Parameters

in	*ne*	number of different events **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:

10.11.2.24 wrap_SensInit1()

```
virtual int wrap_SensInit1 (
            N_Vector * sx,
            N_Vector * sdx,
            const UserData * udata )  [protected], [pure virtual]
```

wrap_SensInit1 initialises the sensitivities at the specified initial timepoint

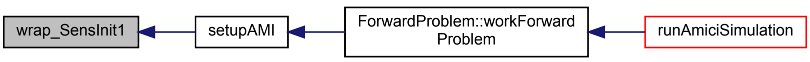
Parameters

in	*sx*	initial state sensitivities **Type**: N_Vector
in	*sdx*	initial derivative state sensitivities (DAE only) **Type**: N_Vector
in	*udata*	initial derivative state sensitivities (DAE only) **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.25 wrap_SetDenseJacFn()

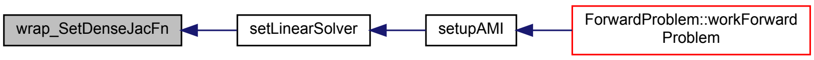
```
virtual int wrap_SetDenseJacFn ( )  [protected], [pure virtual]
```

wrap_SetDenseJacFn sets the dense Jacobian function

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.26 wrap_SetSparseJacFn()

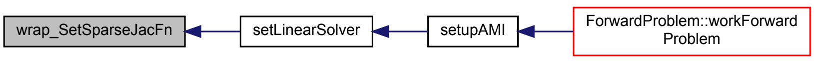
```
virtual int wrap_SetSparseJacFn ( )  [protected], [pure virtual]
```

wrap_SetSparseJacFn sets the sparse Jacobian function

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.27 wrap_SetBandJacFn()

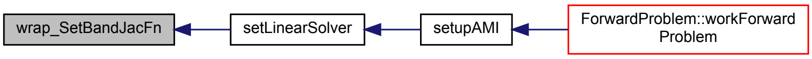
```
virtual int wrap_SetBandJacFn ( )  [protected], [pure virtual]
```

wrap_SetBandJacFn sets the banded Jacobian function

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.28 wrap_SetJacTimesVecFn()

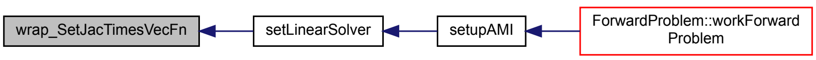
```
virtual int wrap_SetJacTimesVecFn ( )  [protected], [pure virtual]
```

wrap_SetJacTimesVecFn sets the Jacobian vector multiplication function

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.29 wrap_SetDenseJacFnB()

```
virtual int wrap_SetDenseJacFnB (
            int which )  [protected], [pure virtual]
```

wrap_SetDenseJacFn sets the dense Jacobian function

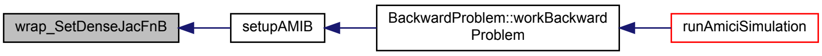
Parameters

in	*which*	identifier of the backwards problem **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.30 wrap_SetSparseJacFnB()

```
virtual int wrap_SetSparseJacFnB (
            int which )  [protected], [pure virtual]
```

wrap_SetSparseJacFn sets the sparse Jacobian function

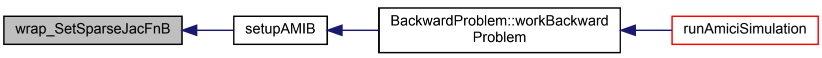
Parameters

in	*which*	identifier of the backwards problem **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.31 wrap_SetBandJacFnB()

```
virtual int wrap_SetBandJacFnB (
            int which )  [protected], [pure virtual]
```

wrap_SetBandJacFn sets the banded Jacobian function

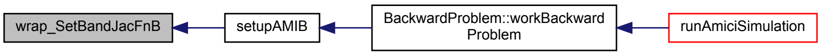
Parameters

in	*which*	identifier of the backwards problem **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.32 wrap_SetJacTimesVecFnB()

```
virtual int wrap_SetJacTimesVecFnB (
            int which ) [protected], [pure virtual]
```

wrap_SetJacTimesVecFn sets the Jacobian vector multiplication function

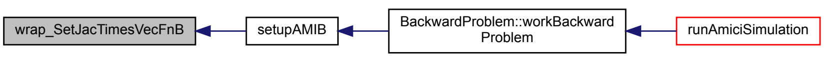
Parameters

in	*which*	identifier of the backwards problem **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.33 wrap_ErrHandlerFn()

```
void wrap_ErrHandlerFn (
            int error_code,
            const char * module,
            const char * function,
            char * msg,
            void * eh_data )  [static], [protected]
```

wrap_ErrHandlerFn extracts diagnosis information from solver memory block and writes them into the return data object for the backward problem

Parameters

in	*error_code*	error identifier **Type**: int
in	*module*	name of the module in which the error occured **Type**: char
in	*function*	name of the function in which the error occured **Type**: char
in	*msg*	error message **Type**: char
in	*eh_data*	unused input

Definition at line 367 of file amici_solver.cpp.

10.11.2.34 AMICreate()

```
virtual void* AMICreate (
            int lmm,
            int iter )  [protected], [pure virtual]
```

AMICreate specifies solver method and initializes solver memory for the forward problem

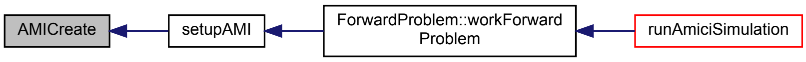
Parameters

in	*lmm*	linear multistep method CV_ADAMS or CV_BDF **Type**: int
in	*iter*	nonlinear solver method CV_NEWTON or CV_FUNCTIONAL **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.35 AMISStolerances()

```
virtual int AMISStolerances (
            double rtol,
            double atol )  [protected], [pure virtual]
```

AMISStolerances sets relative and absolute tolerances for the forward problem

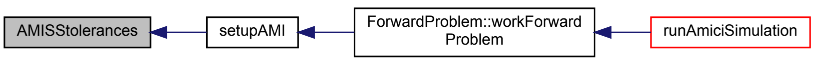
Parameters

in	*rtol*	relative tolerances **Type**: double
in	*atol*	absolute tolerances **Type**: double

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.36 AMISensEEtolerances()

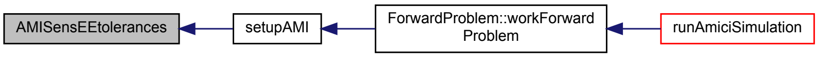
```
virtual int AMISensEEtolerances ( )  [protected], [pure virtual]
```

AMISensEEtolerances activates automatic estimation of tolerances for the forward problem

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.37 AMISetSensErrCon()

```
virtual int AMISetSensErrCon (
            bool error_corr ) [protected], [pure virtual]
```

AMISetSensErrCon specifies whether error control is also enforced for sensitivities for the forward problem

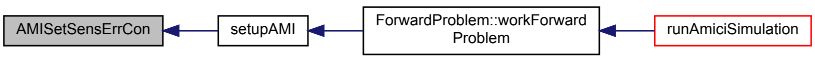
Parameters

in	*error_corr*	activation flag **Type**: bool

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.38 AMISetQuadErrConB()

```
virtual int AMISetQuadErrConB (
            int which,
            bool flag ) [protected], [pure virtual]
```

AMISetSensErrCon specifies whether error control is also enforced for the backward quadrature problem

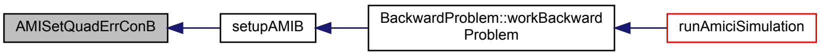
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*flag*	activation flag **Type**: bool

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.39 AMISetErrHandlerFn()

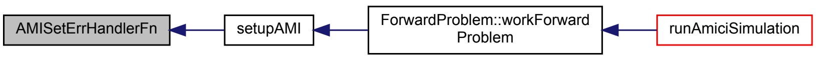
```
virtual int AMISetErrHandlerFn ( ) [protected], [pure virtual]
```

AMISetErrHandlerFn attaches the error handler function (errMsgIdAndTxt) to the solver

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.40 AMISetUserData()

```
virtual int AMISetUserData (
            void * user_data ) [protected], [pure virtual]
```

AMISetUserData attaches the user data object (here this is a TempData and not UserData object) to the forward problem

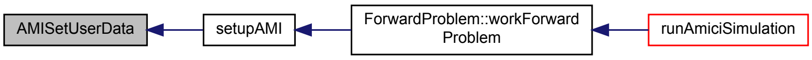
Parameters

in	*user_data*	TempData object, **Type**: TempData

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.41 AMISetUserDataB()

```
virtual int AMISetUserDataB (
            int which,
            void * user_data ) [protected], [pure virtual]
```

AMISetUserDataB attaches the user data object (here this is a TempData and not UserData object) to the backward problem

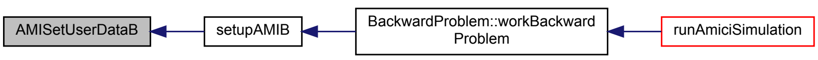
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*user_data*	TempData object, **Type**: TempData

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.42 AMISetMaxNumSteps()

```
virtual int AMISetMaxNumSteps (
            long int mxsteps )  [protected], [pure virtual]
```

AMISetMaxNumSteps specifies the maximum number of steps for the forward problem

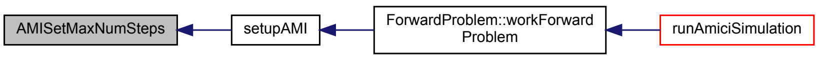
Parameters

in	*mxsteps*	number of steps **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.43 AMISetMaxNumStepsB()

```
virtual int AMISetMaxNumStepsB (
            int which,
            long int mxstepsB )  [protected], [pure virtual]
```

AMISetMaxNumStepsB specifies the maximum number of steps for the forward problem

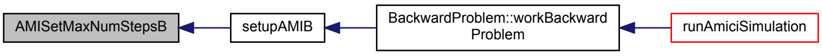
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*mxstepsB*	number of steps **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.44 AMISetStabLimDet()

```
virtual int AMISetStabLimDet (
            int stldet ) [protected], [pure virtual]
```

AMISetStabLimDet activates stability limit detection for the forward problem

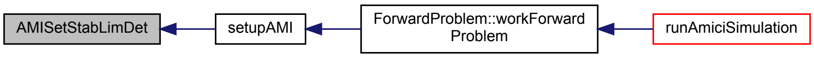
Parameters

in	*stldet*	flag for stability limit detection (TRUE or FALSE) **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.45 AMISetStabLimDetB()

```
virtual int AMISetStabLimDetB (
            int which,
            int stldet ) [protected], [pure virtual]
```

AMISetStabLimDetB activates stability limit detection for the backward problem

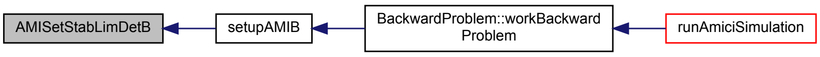
Parameters

`in`	*which*	identifier of the backwards problem **Type**: int
`in`	*stldet*	flag for stability limit detection (TRUE or FALSE) **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.46 AMISetId()

```
virtual int AMISetId (
            Model * model )  [protected], [pure virtual]
```

AMISetId specify algebraic/differential components (DAE only)

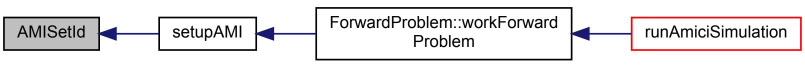
Parameters

`in`	*model*	model specification **Type**: Model

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.47 AMISetSuppressAlg()

```
virtual int AMISetSuppressAlg (
            bool flag ) [protected], [pure virtual]
```

AMISetId deactivates error control for algebraic components (DAE only)

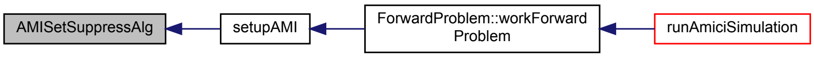
Parameters

in	*flag*	deactivation flag **Type**: bool

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.48 AMISetSensParams()

```
virtual int AMISetSensParams (
            realtype * p,
            realtype * pbar,
            int * plist ) [protected], [pure virtual]
```

AMISetSensParams specifies the scaling and indexes for sensitivity computation

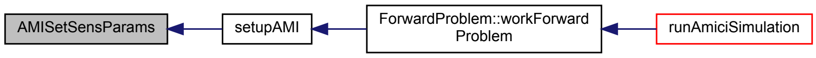
Parameters

in	*p*	paramaters **Type**: realtype
in	*pbar*	parameter scaling constants **Type**: realtype
in	*plist*	parameter index list **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.49 AMIGetDky()

```
virtual int AMIGetDky (
            realtype t,
            int k,
            N_Vector dky )  [protected], [pure virtual]
```

AMIGetDky interpolates the (derivative of the) solution at the requested timepoint

Parameters

in	*t*	timepoint **Type**: realtype
in	*k*	derivative order **Type**: int
out	*dky*	interpolated solution **Type**: N_Vector

Returns

status flag indicating success of execution
Type: int

10.11.2.50 AMIFree()

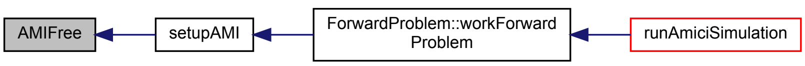
```
virtual void AMIFree ( )  [protected], [pure virtual]
```

AMIFree frees allocation solver memory

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.51 AMIAdjInit()

```
virtual int AMIAdjInit (
            long int steps,
            int interp )  [protected], [pure virtual]
```

AMIAdjInit initializes the adjoint problem

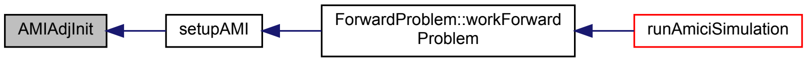
Parameters

in	*steps*	number of integration points between checkpoints **Type**: long int
in	*interp*	interpolation type, can be CV_POLYNOMIAL or CV_HERMITE **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.52 AMICreateB()

```
virtual int AMICreateB (
            int lmm,
            int iter,
            int * which )  [protected], [pure virtual]
```

AMICreateB specifies solver method and initializes solver memory for the backward problem

Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*lmm*	linear multistep method CV_ADAMS or CV_BDF **Type**: int
in	*iter*	nonlinear solver method CV_NEWTON or CV_FUNCTIONAL **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:

10.11.2.53 AMISStolerancesB()

```
virtual int AMISStolerancesB (
            int which,
            realtype relTolB,
            realtype absTolB )  [protected], [pure virtual]
```

AMISStolerancesB sets relative and absolute tolerances for the backward problem

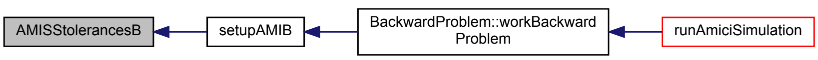
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*relTolB*	relative tolerances **Type**: double
in	*absTolB*	absolute tolerances **Type**: double

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.54 AMIQuadSStolerancesB()

```
virtual int AMIQuadSStolerancesB (
            int which,
            realtype reltolQB,
            realtype abstolQB )  [protected], [pure virtual]
```

AMISStolerancesB sets relative and absolute tolerances for the quadrature backward problem

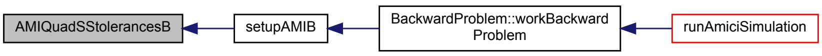
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*reltolQB*	relative tolerances **Type**: double
in	*abstolQB*	absolute tolerances **Type**: double

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.55 AMIDense()

```
virtual int AMIDense (
            int nx )  [protected], [pure virtual]
```

AMIDense attaches a dense linear solver to the forward problem

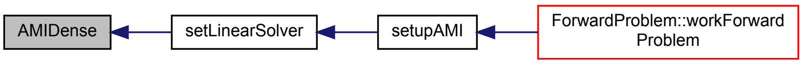
Parameters

in	*nx*	number of state variables **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.56 AMIDenseB()

```
virtual int AMIDenseB (
            int which,
            int nx )  [protected], [pure virtual]
```

AMIDenseB attaches a dense linear solver to the backward problem

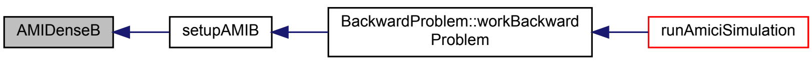
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*nx*	number of state variables **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.57 AMIBand()

```
virtual int AMIBand (
            int nx,
            int ubw,
            int lbw )  [protected], [pure virtual]
```

AMIBand attaches a banded linear solver to the forward problem

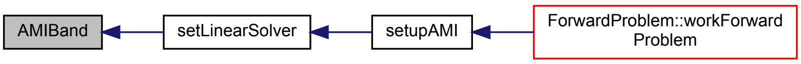
Parameters

in	*nx*	number of state variables **Type**: int
in	*ubw*	upper matrix bandwidth **Type**: int
in	*lbw*	lower matrix bandwidth **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.58 AMIBandB()

```
virtual int AMIBandB (
              int which,
              int nx,
              int ubw,
              int lbw )  [protected], [pure virtual]
```

AMIBandB attaches a banded linear solver to the backward problem

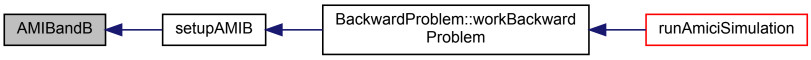
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*nx*	number of state variables **Type**: int
in	*ubw*	upper matrix bandwidth **Type**: int
in	*lbw*	lower matrix bandwidth **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.59 AMIDiag()

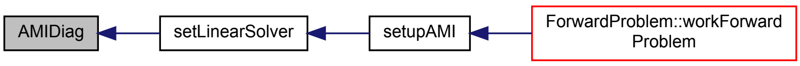
```
virtual int AMIDiag ( )  [protected], [pure virtual]
```

AMIDiag attaches a diagonal linear solver to the forward problem

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.60 AMIDiagB()

```
virtual int AMIDiagB (
            int which )  [protected], [pure virtual]
```

AMIDiagB attaches a diagonal linear solver to the backward problem

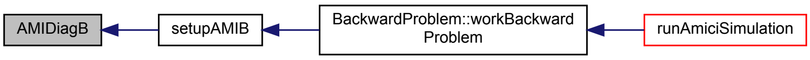
Parameters

in	*which*	identifier of the backwards problem **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.61 AMISpgmr()

```
virtual int AMISpgmr (
            int prectype,
            int maxl )  [protected], [pure virtual]
```

AMIDAMISpgmr attaches a scaled predonditioned GMRES linear solver to the forward problem

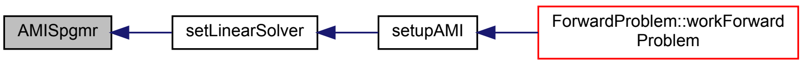
Parameters

in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.62 AMISpgmrB()

```
virtual int AMISpgmrB (
            int which,
            int prectype,
            int maxl )  [protected], [pure virtual]
```

AMIDAMISpgmrB attaches a scaled predonditioned GMRES linear solver to the backward problem

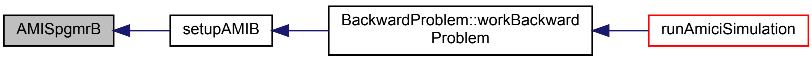
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.63 AMISpbcg()

```
virtual int AMISpbcg (
            int prectype,
            int maxl ) [protected], [pure virtual]
```

AMISpbcg attaches a scaled predonditioned Bi-CGStab linear solver to the forward problem

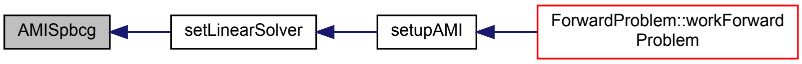
Parameters

in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.64 AMISpbcgB()

```
virtual int AMISpbcgB (
            int which,
            int prectype,
            int maxl )  [protected], [pure virtual]
```

AMISpbcgB attaches a scaled predonditioned Bi-CGStab linear solver to the backward problem

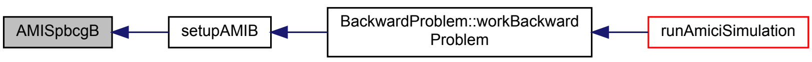
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.65 AMISptfqmr()

```
virtual int AMISptfqmr (
            int prectype,
            int maxl )  [protected], [pure virtual]
```

AMISptfqmr attaches a scaled predonditioned TFQMR linear solver to the forward problem

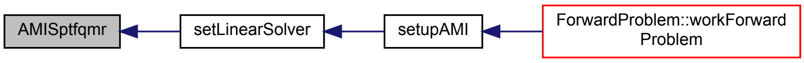
Parameters

in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.66 AMISptfqmrB()

```
virtual int AMISptfqmrB (
            int which,
            int prectype,
            int maxl )  [protected], [pure virtual]
```

AMISptfqmrB attaches a scaled predonditioned TFQMR linear solver to the backward problem

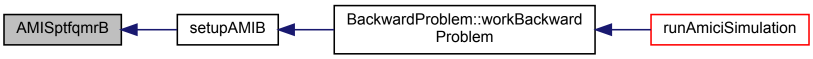
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*prectype*	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH **Type**: int
in	*maxl*	maximum Kryloc subspace dimension **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.67 AMIKLU()

```
virtual int AMIKLU (
            int nx,
            int nnz,
            int sparsetype )  [protected], [pure virtual]
```

AMIKLU attaches a sparse linear solver to the forward problem

Parameters

in	*nx*	number of state variables **Type**: int
in	*nnz*	number of nonzero entries in the jacobian **Type**: int
in	*sparsetype*	sparse storage type, CSC_MAT for column matrix, CSR_MAT for row matrix **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:

10.11.2.68 AMIKLUSetOrdering()

```
virtual int AMIKLUSetOrdering (
            int ordering )  [protected], [pure virtual]
```

AMIKLUSetOrdering sets the ordering for the sparse linear solver of the forward problem

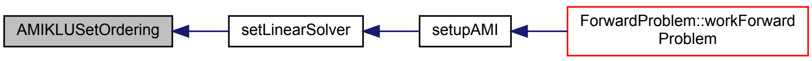
Parameters

in	*ordering*	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.69 AMIKLUSetOrderingB()

```
virtual int AMIKLUSetOrderingB (
            int which,
            int ordering )  [protected], [pure virtual]
```

AMIKLUSetOrderingB sets the ordering for the sparse linear solver of the backward problem

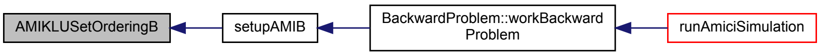
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*ordering*	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.70 AMIKLUB()

```
virtual int AMIKLUB (
            int which,
            int nx,
            int nnz,
            int sparsetype )  [protected], [pure virtual]
```

AMIKLUB attaches a sparse linear solver to the forward problem

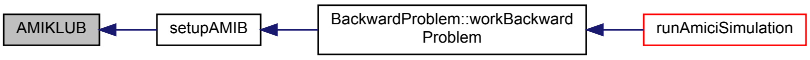
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*nx*	number of state variables **Type**: int
in	*nnz*	number of nonzero entries in the jacobian **Type**: int
in	*sparsetype*	sparse storage type, CSC_MAT for column matrix, CSR_MAT for row matrix **Type**: int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.71 AMIGetNumSteps()

```
virtual int AMIGetNumSteps (
            void * ami_mem,
            long int * numsteps ) [protected], [pure virtual]
```

AMIGetNumSteps reports the number of solver steps

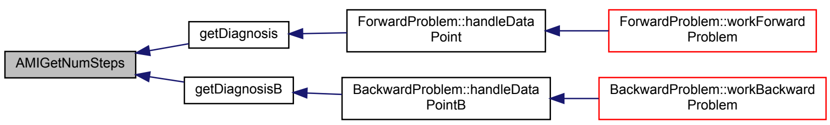
Parameters

in	*ami_mem*	pointer to the solver memory object (can be from forward or backward problem) **Type**: void
out	*numsteps*	output array **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.72 AMIGetNumRhsEvals()

```
virtual int AMIGetNumRhsEvals (
            void * ami_mem,
            long int * numrhsevals ) [protected], [pure virtual]
```

AMIGetNumRhsEvals reports the number of right hand evaluations

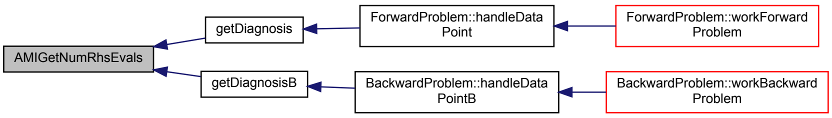
Parameters

in	*ami_mem*	pointer to the solver memory object (can be from forward or backward problem) **Type**: void
out	*numrhsevals*	output array **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.73 AMIGetNumErrTestFails()

```
virtual int AMIGetNumErrTestFails (
            void * ami_mem,
            long int * numerrtestfails )  [protected], [pure virtual]
```

AMIGetNumErrTestFails reports the number of local error test failures

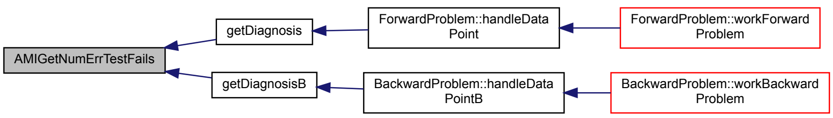
Parameters

in	*ami_mem*	pointer to the solver memory object (can be from forward or backward problem) **Type**: void
out	*numerrtestfails*	output array **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.74 AMIGetNumNonlinSolvConvFails()

```
virtual int AMIGetNumNonlinSolvConvFails (
            void * ami_mem,
            long int * numnonlinsolvconvfails )  [protected], [pure virtual]
```

AMIGetNumNonlinSolvConvFails reports the number of nonlinear convergence failures

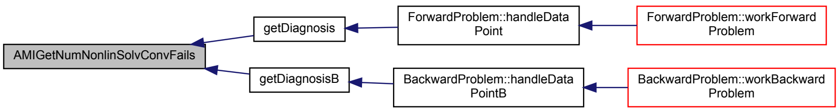
Parameters

in	*ami_mem*	pointer to the solver memory object (can be from forward or backward problem) **Type**: void
out	*numnonlinsolvconvfails*	output array **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.75 AMIGetLastOrder()

```
virtual int AMIGetLastOrder (
            void * ami_mem,
            int * order )  [protected], [pure virtual]
```

AMIGetLastOrder reports the order of the integration method during the last internal step

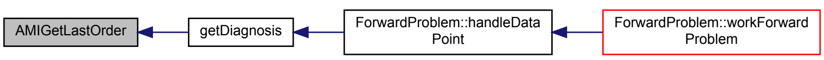
Parameters

in	*ami_mem*	pointer to the solver memory object (can be from forward or backward problem) **Type**: void
out	*order*	output array **Type**: long int

Returns

status flag indicating success of execution
Type: int

Here is the caller graph for this function:



10.11.2.76 AMIGetAdjBmem()

```
virtual void* AMIGetAdjBmem (
            void * ami_mem,
            int which )  [protected], [pure virtual]
```

AMIGetAdjBmem retrieves the solver memory object for the backward problem

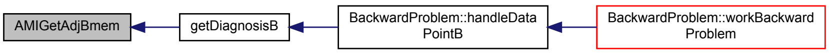
Parameters

in	*which*	identifier of the backwards problem **Type**: int
in	*ami_mem*	pointer to the forward solver memory object **Type**: void

Returns

ami_memB pointer to the backward solver memory object
Type: void

Here is the caller graph for this function:



10.11.2.77 setLinearSolver()

```
int setLinearSolver (
            const UserData * udata,
            Model * model )  [protected]
```

setLinearSolver sets the linear solver

Parameters

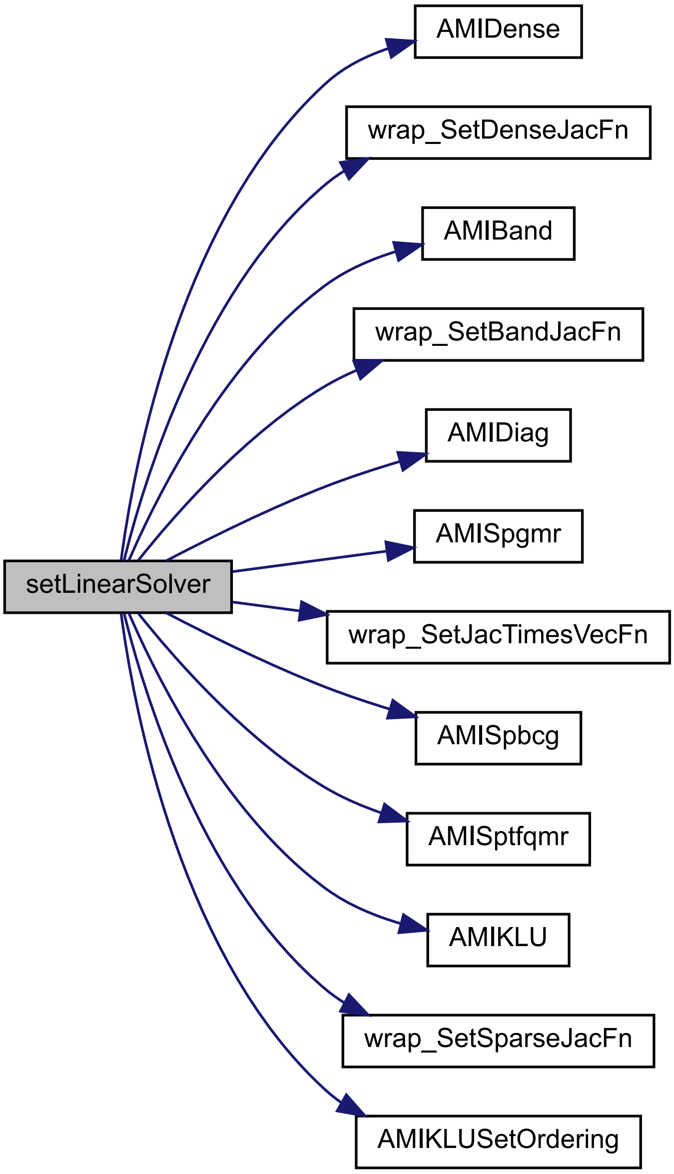
out	*udata*	pointer to the user data object **Type**: UserData
in	*model*	pointer to the model object **Type**: Model

Returns

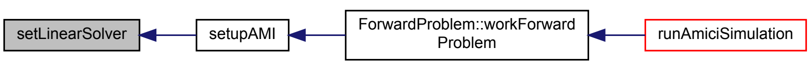
status flag indicating success of execution
Type: int

Definition at line 488 of file amici_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3 Member Data Documentation

10.11.3.1 ami_mem

```
void* ami_mem = nullptr [protected]
```

pointer to ami memory block

Definition at line 702 of file amici_solver.h.

10.12 SteadystateProblem Class Reference

The SteadystateProblem class solves a steady-state problem using Newton's method and falls back to integration on failure.

```
#include <steadystateproblem.h>
```

Static Public Member Functions

- static int workSteadyStateProblem (UserData *udata, TempData *tdata, ReturnData *rdata, Solver *solver, Model *model, int it)
- static int applyNewtonsMethod (UserData *udata, ReturnData *rdata, TempData *tdata, Model *model, NewtonSolver *newtonSolver, int newton_try)
- static void getNewtonOutput (TempData *tdata, ReturnData *rdata, Model *model, int newton_status, double run_time)
- static int getNewtonSimulation (UserData *udata, TempData *tdata, ReturnData *rdata, Solver *solver, Model *model)
- static int linsolveSPBCG (UserData *udata, ReturnData *rdata, TempData *tdata, Model *model, int ntry, int nnewt, N_Vector ns_delta)

10.12.1 Detailed Description

Definition at line 20 of file steadystateproblem.h.

10.12.2 Member Function Documentation

10.12.2.1 workSteadyStateProblem()

```
int workSteadyStateProblem (
            UserData * udata,
            TempData * tdata,
            ReturnData * rdata,
            Solver * solver,
            Model * model,
            int it ) [static]
```

Tries to determine the steady state of the ODE system by a Newton solver, uses forward intergration, if the Newton solver fails, restarts Newton solver, if integration fails. Computes steady state sensitivities

Parameters

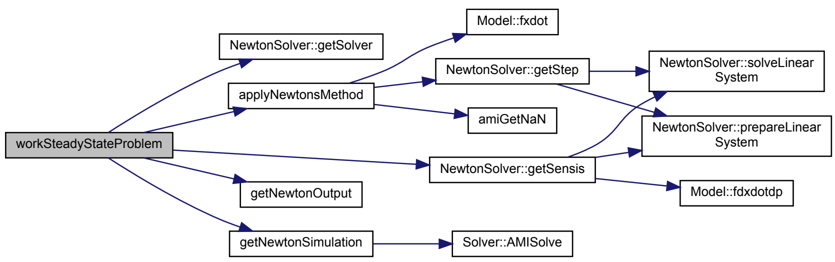
in	*udata*	pointer to the user data object **Type**: UserData
in	*solver*	pointer to the AMICI solver object **Type**: Solver
in	*model*	pointer to the AMICI model object **Type**: Model
in	*it*	integer with the index of the current time step
out	*tdata*	pointer to the temporary data object **Type**: TempData
out	*rdata*	pointer to the return data object **Type**: ReturnData

Returns

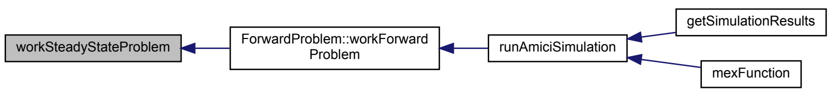
stats integer flag indicating success of the method

Definition at line 16 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.2.2 applyNewtonsMethod()

```
int applyNewtonsMethod (
            UserData * udata,
            ReturnData * rdata,
            TempData * tdata,
            Model * model,
            NewtonSolver * newtonSolver,
            int newton_try )  [static]
```

applyNewtonsMethod applies Newtons method to the current state x to find the steady state Runs the Newton solver iterations and checks for convergence to steady state

Parameters

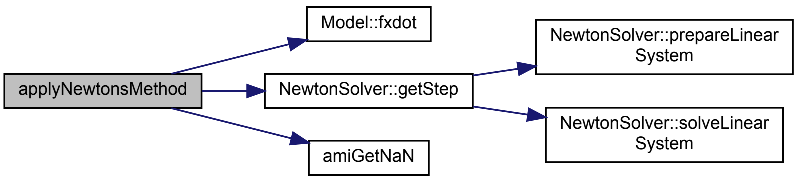
in	*udata*	pointer to the user data object **Type**: UserData
out	*rdata*	pointer to the return data object **Type**: ReturnData
out	*tdata*	pointer to the temporary data object **Type**: TempData
in	*model*	pointer to the AMICI model object **Type**: Model
in	*newtonSolver*	pointer to the NewtonSolver object **Type**: NewtonSolver
in	*newton_try*	integer start number of Newton solver (1 or 2)

Returns

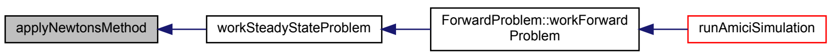
stats integer flag indicating success of the method

Definition at line 86 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.2.3 getNewtonOutput()

```
void getNewtonOutput (
            TempData * tdata,
            ReturnData * rdata,
            Model * model,
            int newton_status,
            double run_time )  [static]
```

Stores output of workSteadyStateProblem in return data

Parameters

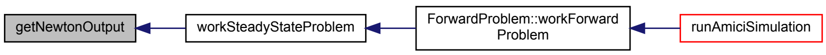
in	*tdata*	pointer to the temporary data object **Type**: UserData
in	*model*	pointer to the AMICI model object **Type**: Model
in	*newton_status*	integer flag indicating when a steady state was found
in	*run_time*	double coputation time of the solver in milliseconds
out	*rdata*	pointer to the return data object **Type**: ReturnData

Returns

stats integer flag indicating success of the method

Definition at line 236 of file steadystateproblem.cpp.

Here is the caller graph for this function:



10.12.2.4 getNewtonSimulation()

```
int getNewtonSimulation (
            UserData * udata,
            TempData * tdata,
            ReturnData * rdata,
            Solver * solver,
            Model * model )  [static]
```

Forward simulation is launched, if Newton solver fails in first try

Parameters

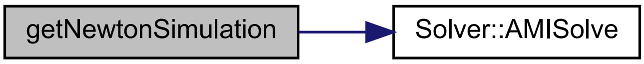
in	*udata*	pointer to the user data object **Type**: UserData
in	*solver*	pointer to the AMICI solver object **Type**: Solver
in	*model*	pointer to the AMICI model object **Type**: Model
out	*tdata*	pointer to the temporary data object **Type**: TempData
out	*rdata*	pointer to the return data object **Type**: ReturnData

Returns

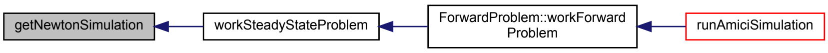
stats integer flag indicating success of the method

Definition at line 271 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.2.5 linsolveSPBCG()

```
int linsolveSPBCG (
            UserData * udata,
            ReturnData * rdata,
            TempData * tdata,
            Model * model,
            int ntry,
            int nnewt,
            N_Vector ns_delta )  [static]
```

Iterative linear solver created from SPILS BiCG-Stab. Solves the linear system within each Newton step if iterative solver is chosen.

Parameters

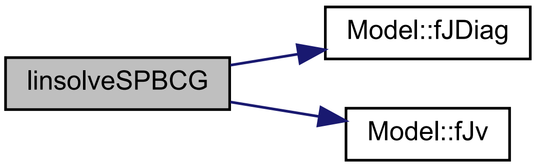
in	*udata*	pointer to the user data object **Type**: UserData
in	*model*	pointer to the AMICI model object **Type**: Model
in	*ntry*	integer newton_try integer start number of Newton solver (1 or 2)
in	*nnewt*	integer number of current Newton step
in	*ns_delta*	???
out	*tdata*	pointer to the temporary data object **Type**: TempData
out	*rdata*	pointer to the return data object **Type**: ReturnData

Returns

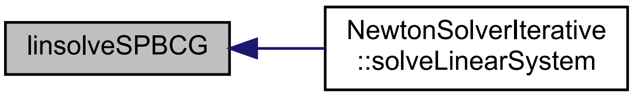
stats integer flag indicating success of the method

Definition at line 335 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

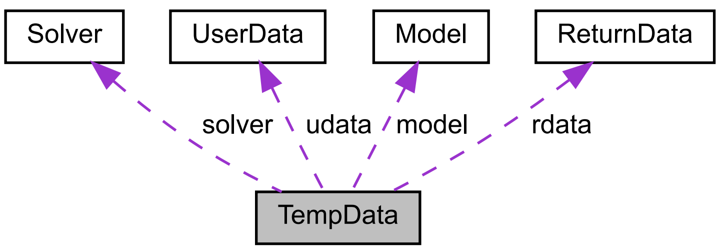


10.13 TempData Class Reference

struct that provides temporary storage for different variables

`#include <tdata.h>`

Collaboration diagram for TempData:



Public Member Functions

- TempData (const UserData ∗udata, Model ∗model, ReturnData ∗rdata)

Public Attributes

- realtype ∗ p = nullptr
- realtype t
- N_Vector x
- N_Vector x_old
- N_Vector ∗ x_disc
- N_Vector ∗ xdot_disc
- N_Vector ∗ xdot_old_disc
- N_Vector dx
- N_Vector dx_old
- N_Vector xdot
- N_Vector xdot_old
- N_Vector xB
- N_Vector xB_old
- N_Vector dxB
- N_Vector xQB
- N_Vector xQB_old
- N_Vector ∗ sx
- N_Vector ∗ sdx
- DlsMat Jtmp
- realtype ∗ llhS0
- realtype ∗ Jy
- realtype ∗ dJydp
- realtype ∗ dJydy
- realtype ∗ dJydsigma
- realtype ∗ dJydx
- realtype ∗ Jz
- realtype ∗ dJzdp
- realtype ∗ dJzdx
- realtype ∗ dJzdz
- realtype ∗ dJzdsigma
- realtype ∗ dJrzdz
- realtype ∗ dJrzdsigma
- realtype ∗ dzdx
- realtype ∗ dzdp
- realtype ∗ drzdx
- realtype ∗ drzdp
- realtype ∗ dydp
- realtype ∗ dydx
- realtype ∗ yS0
- realtype ∗ sigmay
- realtype ∗ dsigmaydp
- realtype ∗ sigmaz
- realtype ∗ dsigmazdp
- int ∗ rootsfound
- int ∗ rootidx
- int ∗ nroots
- realtype ∗ rootvals
- realtype ∗ h

- realtype * h_udata
- realtype * deltax
- realtype * deltasx
- realtype * deltaxB
- realtype * deltaqB
- int which
- realtype * discs
- realtype * irdiscs
- SlsMat J = NULL
- realtype * dxdotdp = NULL
- realtype * w = NULL
- realtype * dwdx = NULL
- realtype * dwdp = NULL
- realtype * M = NULL
- realtype * dfdx = NULL
- realtype * stau = NULL
- int nplist
- int iroot = 0
- booleantype nan_dxdotdp = false
- booleantype nan_J = false
- booleantype nan_JDiag = false
- booleantype nan_JSparse = false
- booleantype nan_xdot = false
- booleantype nan_xBdot = false
- booleantype nan_qBdot = false
- const UserData * udata
- Model * model
- ReturnData * rdata
- Solver * solver = nullptr

10.13.1 Detailed Description

Definition at line 17 of file tdata.h.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 TempData()

```
TempData (
            const UserData * udata,
            Model * model,
            ReturnData * rdata )
```

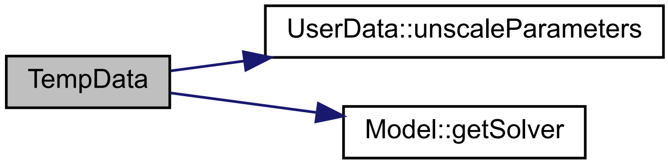
Default constructor

Parameters

in	*udata*	pointer to the user data struct **Type**: UserData
in	*model*	pointer to model specification object **Type**: Model
in	*rdata*	pointer to the return data struct **Type**: ReturnData

Definition at line 8 of file tdata.cpp.

Here is the call graph for this function:



10.13.3 Member Data Documentation

10.13.3.1 p

```
realtype* p = nullptr
```

parameter array, unscaled

Definition at line 24 of file tdata.h.

10.13.3.2 t

```
realtype t
```

current time

Definition at line 27 of file tdata.h.

10.13.3.3 x

```
N_Vector x
```

state vector

Definition at line 30 of file tdata.h.

10.13.3.4 x_old

```
N_Vector x_old
```

old state vector

Definition at line 32 of file tdata.h.

10.13.3.5 x_disc

```
N_Vector* x_disc
```

array of state vectors at discontinuities

Definition at line 34 of file tdata.h.

10.13.3.6 xdot_disc

```
N_Vector* xdot_disc
```

array of differential state vectors at discontinuities

Definition at line 36 of file tdata.h.

10.13.3.7 xdot_old_disc

```
N_Vector* xdot_old_disc
```

array of old differential state vectors at discontinuities

Definition at line 38 of file tdata.h.

10.13.3.8 dx

```
N_Vector dx
```

differential state vector

Definition at line 40 of file tdata.h.

10.13.3.9 dx_old

```
N_Vector dx_old
```

old differential state vector

Definition at line 42 of file tdata.h.

10.13.3.10 xdot

```
N_Vector xdot
```

time derivative state vector

Definition at line 44 of file tdata.h.

10.13.3.11 xdot_old

```
N_Vector xdot_old
```

old time derivative state vector

Definition at line 46 of file tdata.h.

10.13.3.12 xB

```
N_Vector xB
```

adjoint state vector

Definition at line 48 of file tdata.h.

10.13.3.13 xB_old

```
N_Vector xB_old
```

old adjoint state vector

Definition at line 50 of file tdata.h.

10.13.3.14 dxB

`N_Vector dxB`

differential adjoint state vector

Definition at line 52 of file tdata.h.

10.13.3.15 xQB

`N_Vector xQB`

quadrature state vector

Definition at line 54 of file tdata.h.

10.13.3.16 xQB_old

`N_Vector xQB_old`

old quadrature state vector

Definition at line 56 of file tdata.h.

10.13.3.17 sx

`N_Vector* sx`

sensitivity state vector array

Definition at line 58 of file tdata.h.

10.13.3.18 sdx

`N_Vector* sdx`

differential sensitivity state vector array

Definition at line 60 of file tdata.h.

10.13.3.19 Jtmp

```
DlsMat Jtmp
```

Jacobian

Definition at line 62 of file tdata.h.

10.13.3.20 llhS0

```
realtype* llhS0
```

parameter derivative of likelihood array

Definition at line 65 of file tdata.h.

10.13.3.21 Jy

```
realtype* Jy
```

data likelihood

Definition at line 67 of file tdata.h.

10.13.3.22 dJydp

```
realtype* dJydp
```

parameter derivative of data likelihood

Definition at line 69 of file tdata.h.

10.13.3.23 dJydy

```
realtype* dJydy
```

observable derivative of data likelihood

Definition at line 71 of file tdata.h.

10.13.3.24 dJydsigma

```
realtype* dJydsigma
```

observable sigma derivative of data likelihood

Definition at line 73 of file tdata.h.

10.13.3.25 dJydx

```
realtype* dJydx
```

state derivative of data likelihood

Definition at line 75 of file tdata.h.

10.13.3.26 Jz

```
realtype* Jz
```

event likelihood

Definition at line 77 of file tdata.h.

10.13.3.27 dJzdp

```
realtype* dJzdp
```

parameter derivative of event likelihood

Definition at line 79 of file tdata.h.

10.13.3.28 dJzdx

```
realtype* dJzdx
```

state derivative of event likelihood

Definition at line 81 of file tdata.h.

10.13.3.29 dJzdz

`realtype* dJzdz`

event ouput derivative of event likelihood

Definition at line 83 of file tdata.h.

10.13.3.30 dJzdsigma

`realtype* dJzdsigma`

event sigma derivative of event likelihood

Definition at line 85 of file tdata.h.

10.13.3.31 dJrzdz

`realtype* dJrzdz`

event ouput derivative of event likelihood at final timepoint

Definition at line 87 of file tdata.h.

10.13.3.32 dJrzdsigma

`realtype* dJrzdsigma`

event sigma derivative of event likelihood at final timepoint

Definition at line 89 of file tdata.h.

10.13.3.33 dzdx

`realtype* dzdx`

state derivative of event output

Definition at line 91 of file tdata.h.

10.13.3.34 dzdp

```
realtype* dzdp
```

parameter derivative of event output

Definition at line 93 of file tdata.h.

10.13.3.35 drzdx

```
realtype* drzdx
```

state derivative of event timepoint

Definition at line 95 of file tdata.h.

10.13.3.36 drzdp

```
realtype* drzdp
```

parameter derivative of event timepoint

Definition at line 97 of file tdata.h.

10.13.3.37 dydp

```
realtype* dydp
```

parameter derivative of observable

Definition at line 99 of file tdata.h.

10.13.3.38 dydx

```
realtype* dydx
```

state derivative of observable

Definition at line 101 of file tdata.h.

10.13.3.39 yS0

```
realtype* yS0
```

initial sensitivity of observable

Definition at line 103 of file tdata.h.

10.13.3.40 sigmay

```
realtype* sigmay
```

data standard deviation

Definition at line 105 of file tdata.h.

10.13.3.41 dsigmaydp

```
realtype* dsigmaydp
```

parameter derivative of data standard deviation

Definition at line 107 of file tdata.h.

10.13.3.42 sigmaz

```
realtype* sigmaz
```

event standard deviation

Definition at line 109 of file tdata.h.

10.13.3.43 dsigmazdp

```
realtype* dsigmazdp
```

parameter derivative of event standard deviation

Definition at line 111 of file tdata.h.

10.13.3.44 rootsfound

```
int* rootsfound
```

array of flags indicating which root has beend found

array of length nr with the indices of the user functions gi found to have a root. For i = 0, . . . ,nr 1 if gi has a root, and = 0 if not.

Definition at line 118 of file tdata.h.

10.13.3.45 rootidx

```
int* rootidx
```

array of index which root has been found

Definition at line 120 of file tdata.h.

10.13.3.46 nroots

```
int* nroots
```

array of number of found roots for a certain event type

Definition at line 122 of file tdata.h.

10.13.3.47 rootvals

```
realtype* rootvals
```

array of values of the root function

Definition at line 124 of file tdata.h.

10.13.3.48 h

```
realtype* h
```

temporary rootval storage to check crossing in secondary event

Definition at line 126 of file tdata.h.

10.13.3.49 h_udata

```
realtype* h_udata
```

flag indicating whether a certain heaviside function should be active or not Moved from UserData to TempData; TODO: better naming

Definition at line 131 of file tdata.h.

10.13.3.50 deltax

```
realtype* deltax
```

change in x

Definition at line 134 of file tdata.h.

10.13.3.51 deltasx

```
realtype* deltasx
```

change in sx

Definition at line 136 of file tdata.h.

10.13.3.52 deltaxB

```
realtype* deltaxB
```

change in xB

Definition at line 138 of file tdata.h.

10.13.3.53 deltaqB

```
realtype* deltaqB
```

change in qB

Definition at line 140 of file tdata.h.

10.13.3.54 which

```
int which
```

integer for indexing of backwards problems

Definition at line 143 of file tdata.h.

10.13.3.55 discs

```
realtype* discs
```

array containing the time-points of discontinuities

Definition at line 146 of file tdata.h.

10.13.3.56 irdiscs

```
realtype* irdiscs
```

array containing the index of discontinuities

Definition at line 148 of file tdata.h.

10.13.3.57 J

```
SlsMat J = NULL
```

tempory storage of Jacobian data across functions

Definition at line 151 of file tdata.h.

10.13.3.58 dxdotdp

```
realtype* dxdotdp = NULL
```

tempory storage of dxdotdp data across functions

Definition at line 153 of file tdata.h.

10.13.3.59 w

```
realtype* w = NULL
```

tempory storage of w data across functions

Definition at line 155 of file tdata.h.

10.13.3.60 dwdx

```
realtype* dwdx = NULL
```

tempory storage of dwdx data across functions

Definition at line 157 of file tdata.h.

10.13.3.61 dwdp

```
realtype* dwdp = NULL
```

tempory storage of dwdp data across functions

Definition at line 159 of file tdata.h.

10.13.3.62 M

```
realtype* M = NULL
```

tempory storage of M data across functions

Definition at line 161 of file tdata.h.

10.13.3.63 dfdx

```
realtype* dfdx = NULL
```

tempory storage of dfdx data across functions

Definition at line 163 of file tdata.h.

10.13.3.64 stau

```
realtype* stau = NULL
```

tempory storage of stau data across functions

Definition at line 165 of file tdata.h.

10.13.3.65 nplist

```
int nplist
```

number of parameters, copied from udata, necessary for deallocation

Definition at line 168 of file tdata.h.

10.13.3.66 iroot

```
int iroot = 0
```

current root index, will be increased during the forward solve and decreased during backward solve

Definition at line 172 of file tdata.h.

10.13.3.67 nan_dxdotdp

```
booleantype nan_dxdotdp = false
```

flag indicating whether a NaN in dxdotdp has been reported

Definition at line 175 of file tdata.h.

10.13.3.68 nan_J

```
booleantype nan_J = false
```

flag indicating whether a NaN in J has been reported

Definition at line 177 of file tdata.h.

10.13.3.69 nan_JDiag

```
booleantype nan_JDiag = false
```

flag indicating whether a NaN in JDiag has been reported

Definition at line 179 of file tdata.h.

10.13.3.70 nan_JSparse

```
booleantype nan_JSparse = false
```

flag indicating whether a NaN in JSparse has been reported

Definition at line 181 of file tdata.h.

10.13.3.71 nan_xdot

```
booleantype nan_xdot = false
```

flag indicating whether a NaN in xdot has been reported

Definition at line 183 of file tdata.h.

10.13.3.72 nan_xBdot

```
booleantype nan_xBdot = false
```

flag indicating whether a NaN in xBdot has been reported

Definition at line 185 of file tdata.h.

10.13.3.73 nan_qBdot

```
booleantype nan_qBdot = false
```

flag indicating whether a NaN in qBdot has been reported

Definition at line 187 of file tdata.h.

10.13.3.74 udata

```
const UserData* udata
```

attached UserData object

Definition at line 190 of file tdata.h.

10.13.3.75 model

```
Model* model
```

attached Model object

Definition at line 192 of file tdata.h.

10.13.3.76 rdata

```
ReturnData* rdata
```

attached ReturnData object

Definition at line 194 of file tdata.h.

10.13.3.77 solver

```
Solver* solver = nullptr
```

attached Solver object

Definition at line 196 of file tdata.h.

10.14 UserData Class Reference

struct that stores all user provided data

```
#include <udata.h>
```

Public Member Functions

- UserData ()

  Default constructor for testing and serialization.
- int unscaleParameters (const Model *model, double *bufferUnscaled) const
- void print ()

Public Attributes

- int nmaxevent
- double * qpositivex
- int * plist
- int nplist
- int nt
- double * p
- double * k
- AMICI_parameter_scaling pscale
- double tstart
- double * ts
- double * pbar
- double * xbar
- AMICI_sensi_order sensi
- double atol
- double rtol
- int maxsteps
- int newton_maxsteps
- int newton_maxlinsteps
- int newton_preeq
- int newton_precon
- int ism
- AMICI_sensi_meth sensi_meth
- int linsol
- int interpType
- int lmm
- int iter
- booleantype stldet
- double * x0data
- double * sx0data
- int ordering

Protected Member Functions

- void init ()

10.14.1 Detailed Description

Definition at line 10 of file udata.h.

10.14.2 Member Function Documentation

10.14.2.1 unscaleParameters()

```
int unscaleParameters (
            const Model * model,
            double * bufferUnscaled ) const
```

unscaleParameters removes parameter scaling according to the parameter scaling in pscale

Parameters

in	*model*	pointer to model specification object **Type**: Model
out	*bufferUnscaled*	unscaled parameters are written to the array **Type**: double

Returns

status flag indicating success of execution
Type: int

Definition at line 8 of file udata.cpp.

Here is the caller graph for this function:

10.14.2.2 print()

```
void print ( )
```

function to print the contents of the UserData object

Definition at line 92 of file udata.cpp.

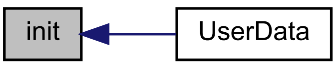
10.14.2.3 init()

```
void init ( ) [protected]
```

function to initialize the contents of the UserData object

Definition at line 58 of file udata.cpp.

Here is the caller graph for this function:



10.14.3 Member Data Documentation

10.14.3.1 nmaxevent

```
int nmaxevent
```

maximal number of events to track

Definition at line 25 of file udata.h.

10.14.3.2 qpositivex

```
double* qpositivex
```

positivity flag

Definition at line 28 of file udata.h.

10.14.3.3 plist

```
int* plist
```

parameter selection and reordering

Definition at line 31 of file udata.h.

10.14.3.4 nplist

```
int nplist
```

number of parameters in plist

Definition at line 33 of file udata.h.

10.14.3.5 nt

```
int nt
```

number of timepoints

Definition at line 36 of file udata.h.

10.14.3.6 p

```
double* p
```

parameter array

Definition at line 39 of file udata.h.

10.14.3.7 k

```
double* k
```

constants array

Definition at line 42 of file udata.h.

10.14.3.8 pscale

```
AMICI_parameter_scaling pscale
```

parameter transformation of p

Definition at line 45 of file udata.h.

10.14.3.9 tstart

```
double tstart
```

starting time

Definition at line 48 of file udata.h.

10.14.3.10 ts

```
double* ts
```

timepoints

Definition at line 50 of file udata.h.

10.14.3.11 pbar

```
double* pbar
```

scaling of parameters

Definition at line 53 of file udata.h.

10.14.3.12 xbar

```
double* xbar
```

scaling of states

Definition at line 55 of file udata.h.

10.14.3.13 sensi

```
AMICI_sensi_order sensi
```

flag indicating whether sensitivities are supposed to be computed

Definition at line 58 of file udata.h.

10.14.3.14 atol

```
double atol
```

absolute tolerances for integration

Definition at line 60 of file udata.h.

10.14.3.15 rtol

```
double rtol
```

relative tolerances for integration

Definition at line 62 of file udata.h.

10.14.3.16 maxsteps

`int maxsteps`

maximum number of allowed integration steps

Definition at line 64 of file udata.h.

10.14.3.17 newton_maxsteps

`int newton_maxsteps`

maximum number of allowed Newton steps for steady state computation

Definition at line 67 of file udata.h.

10.14.3.18 newton_maxlinsteps

`int newton_maxlinsteps`

maximum number of allowed linear steps per Newton step for steady state computation

Definition at line 70 of file udata.h.

10.14.3.19 newton_preeq

`int newton_preeq`

Preequilibration of model via NEwton solver?

Definition at line 72 of file udata.h.

10.14.3.20 newton_precon

`int newton_precon`

Which preconditioner is to be used in the case of iterative linear Newton solvers

Definition at line 75 of file udata.h.

10.14.3.21 ism

```
int ism
```

internal sensitivity method

a flag used to select the sensitivity solution method. Its value can be CV SIMULTANEOUS or CV STAGGERED. Only applies for Forward Sensitivities.

Definition at line 82 of file udata.h.

10.14.3.22 sensi_meth

```
AMICI_sensi_meth sensi_meth
```

method for sensitivity computation

Definition at line 85 of file udata.h.

10.14.3.23 linsol

```
int linsol
```

linear solver specification

Definition at line 88 of file udata.h.

10.14.3.24 interpType

```
int interpType
```

interpolation type

specifies the interpolation type for the forward problem solution which is then used for the backwards problem. can be either CV_POLYNOMIAL or CV_HERMITE

Definition at line 96 of file udata.h.

10.14.3.25 lmm

```
int lmm
```

linear multistep method

specifies the linear multistep method and may be one of two possible values: CV ADAMS or CV BDF.

Definition at line 103 of file udata.h.

10.14.3.26 iter

`int iter`

nonlinear solver

specifies the type of nonlinear solver iteration and may be either CV NEWTON or CV FUNCTIONAL.

Definition at line 110 of file udata.h.

10.14.3.27 stldet

`booleantype stldet`

flag controlling stability limit detection

Definition at line 113 of file udata.h.

10.14.3.28 x0data

`double* x0data`

state initialisation

Definition at line 116 of file udata.h.

10.14.3.29 sx0data

`double* sx0data`

sensitivity initialisation

Definition at line 119 of file udata.h.

10.14.3.30 ordering

`int ordering`

state ordering

Definition at line 122 of file udata.h.

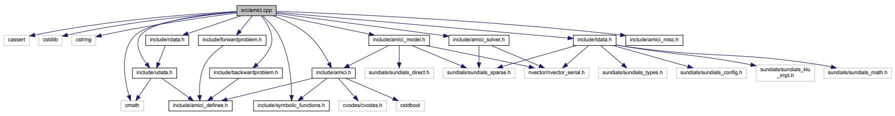
11 File Documentation

11.1 src/amici.cpp File Reference

core routines for integration

```
#include <cassert>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include "include/amici_model.h"
#include "include/amici_solver.h"
#include "include/backwardproblem.h"
#include "include/forwardproblem.h"
#include "include/rdata.h"
#include "include/tdata.h"
#include "include/udata.h"
#include <include/amici.h>
#include <include/amici_misc.h>
#include <include/symbolic_functions.h>
```

Include dependency graph for amici.cpp:



Macros

- #define _USE_MATH_DEFINES
- #define M_PI 3.14159265358979323846

Functions

- int runAmiciSimulation (UserData ∗udata, const ExpData ∗edata, ReturnData ∗rdata, Model ∗model)
- void printErrMsgIdAndTxt (const char ∗identifier, const char ∗msg,...)
- void printWarnMsgIdAndTxt (const char ∗identifier, const char ∗msg,...)

Variables

- msgIdAndTxtFp errMsgIdAndTxt = &printErrMsgIdAndTxt
- msgIdAndTxtFp warnMsgIdAndTxt = &printWarnMsgIdAndTxt

11.1.1 Macro Definition Documentation

11.1.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

MS definition of PI and other constants

Definition at line 10 of file amici.cpp.

11.1.1.2 M_PI

```
#define M_PI 3.14159265358979323846
```

define PI if we still have no definition

Definition at line 14 of file amici.cpp.

11.1.2 Function Documentation

11.1.2.1 runAmiciSimulation()

```
int runAmiciSimulation (
            UserData * udata,
            const ExpData * edata,
            ReturnData * rdata,
            Model * model )
```

runAmiciSimulation is the core integration routine. It initializes the solver and temporary storage in tdata and runs the forward and backward problem.

Parameters

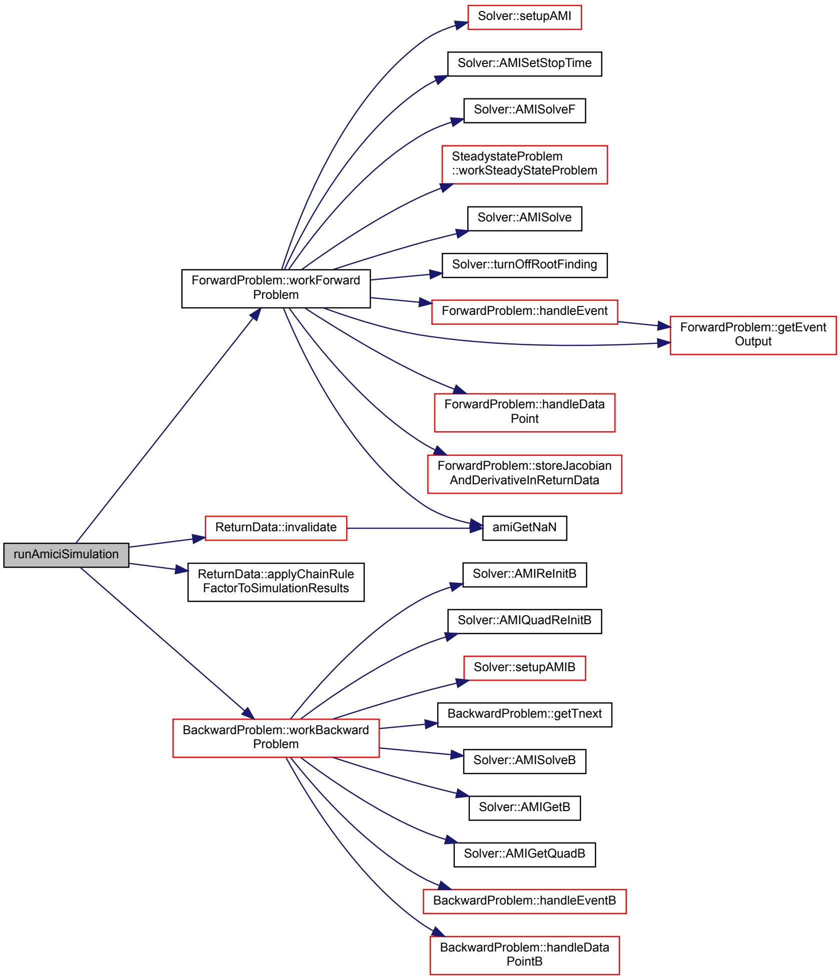
in	*udata*	pointer to user data object **Type**: UserData
in	*edata*	pointer to experimental data object **Type**: ExpData
in	*rdata*	pointer to return data object **Type**: ReturnData
in	*model*	pointer to model specification object **Type**: Model

Returns

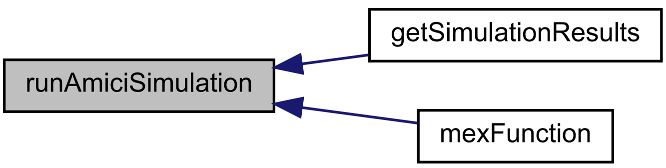
status status flag indicating (un)successful execution
Type: int

Definition at line 43 of file amici.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



11.1.2.2 printErrMsgIdAndTxt()

```
void printErrMsgIdAndTxt (
            const char * identifier,
            const char * msg,
             ... )
```

printErrMsgIdAndTxt prints a specified error message associated to the specified identifier

Parameters

in	*identifier*	error identifier **Type**: char
in	*msg*	error message **Type**: char

Returns

void

Definition at line 81 of file amici.cpp.

11.1.2.3 printWarnMsgIdAndTxt()

```
void printWarnMsgIdAndTxt (
            const char * identifier,
            const char * msg,
             ... )
```

printErrMsgIdAndTxt prints a specified warning message associated to the specified identifier

Parameters

in	*identifier*	warning identifier **Type**: char
in	*msg*	warning message **Type**: char

Returns

void

Definition at line 92 of file amici.cpp.

11.1.3 Variable Documentation

11.1.3.1 errMsgIdAndTxt

```
msgIdAndTxtFp errMsgIdAndTxt = &printErrMsgIdAndTxt
```

errMsgIdAndTxt is a function pointer for printErrMsgIdAndTxt

Definition at line 29 of file amici.cpp.

11.1.3.2 warnMsgIdAndTxt

```
msgIdAndTxtFp warnMsgIdAndTxt = &printWarnMsgIdAndTxt
```

warnMsgIdAndTxt is a function pointer for printWarnMsgIdAndTxt

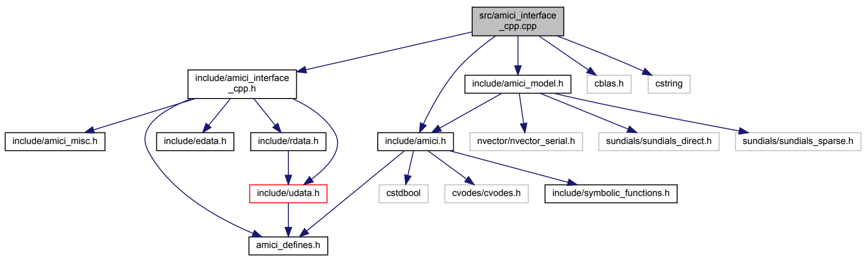
Definition at line 31 of file amici.cpp.

11.2 src/amici_interface_cpp.cpp File Reference

core routines for cpp interface

```
#include "include/amici_interface_cpp.h"
#include "include/amici.h"
#include <include/amici_model.h>
#include <cblas.h>
#include <cstring>
```

Include dependency graph for amici_interface_cpp.cpp:



Functions

- ReturnData ∗ getSimulationResults (Model ∗model, UserData ∗udata, const ExpData ∗edata)
- void amici_dgemm (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, AMICI_BLAS_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, const double beta, double ∗C, const int ldc)
- void amici_dgemv (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)

11.2.1 Function Documentation

11.2.1.1 getSimulationResults()

```
ReturnData* getSimulationResults (
            Model * model,
            UserData * udata,
            const ExpData * edata )
```

getSimulationResults is the core cpp interface function. It initializes the model and return data and then calls the core simulation routine.

Parameters

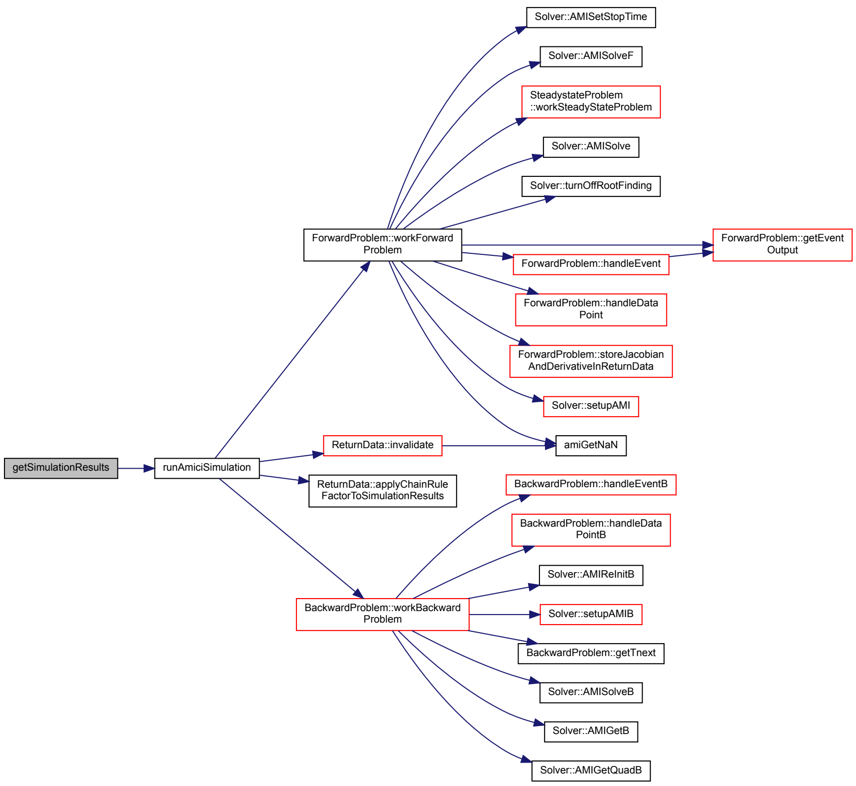
in	*model*	pointer to the model object, this is necessary to perform dimension checks **Type**: Model
in	*udata*	pointer to user data object **Type**: UserData
in	*edata*	pointer to experimental data object **Type**: ExpData

Returns

rdata pointer to return data object
Type: ReturnData

Definition at line 31 of file amici_interface_cpp.cpp.

Here is the call graph for this function:



11.2.1.2 amici_dgemm()

```
void amici_dgemm (
            AMICI_BLAS_LAYOUT layout,
            AMICI_BLAS_TRANSPOSE TransA,
            AMICI_BLAS_TRANSPOSE TransB,
            const int M,
            const int N,
            const int K,
            const double alpha,
            const double * A,
            const int lda,
            const double * B,
            const int ldb,
            const double beta,
            double * C,
            const int ldc )
```

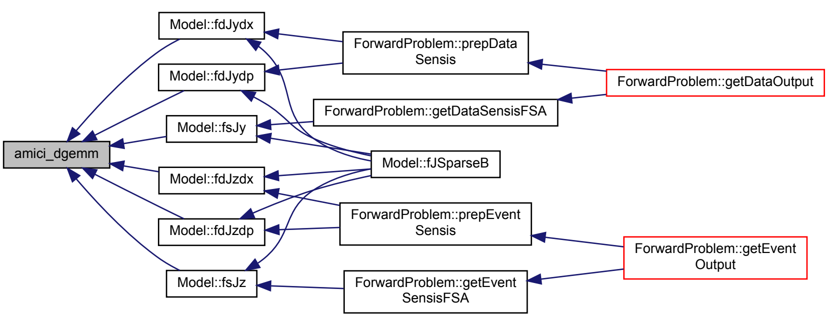
amici_dgemm provides an interface to the blas matrix matrix multiplication routine dgemm. This routines computes C = alpha∗A∗B + beta∗C with A: [MxK] B:[KxN] C:[MxN]

Parameters

in	*layout*	can be AMICI_BLAS_ColMajor or AMICI_BLAS_RowMajor.
in	*TransA*	flag indicating whether A should be transposed before multiplication
in	*TransB*	flag indicating whether B should be transposed before multiplication
in	*M*	number of rows in A/C
in	*N*	number of columns in B/C
in	*K*	number of rows in B, number of columns in A
in	*alpha*	coefficient alpha
in	*A*	matrix A
in	*lda*	leading dimension of A (m or k)
in	*B*	matrix B
in	*ldb*	leading dimension of B (k or n)
in	*beta*	coefficient beta
in,out	*C*	matrix C
in	*ldc*	leading dimension of C (m or n)

Definition at line 60 of file amici_interface_cpp.cpp.

Here is the caller graph for this function:



11.2.1.3 amici_dgemv()

```
void amici_dgemv (
            AMICI_BLAS_LAYOUT layout,
            AMICI_BLAS_TRANSPOSE TransA,
            const int M,
            const int N,
            const double alpha,
            const double * A,
            const int lda,
            const double * X,
            const int incX,
            const double beta,
            double * Y,
            const int incY )
```

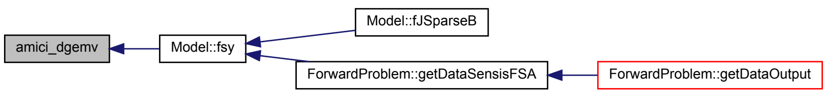
amici_dgemm provides an interface to the blas matrix vector multiplication routine dgemv. This routines computes y = alpha∗A∗x + beta∗y with A: [MxK] B:[KxN] C:[MxN]

Parameters

in	*layout*	can be AMICI_BLAS_ColMajor or AMICI_BLAS_RowMajor.
in	*TransA*	flag indicating whether A should be transposed before multiplication
in	*M*	number of rows in A
in	*N*	number of columns in A
in	*alpha*	coefficient alpha
in	*A*	matrix A
in	*lda*	leading dimension of A (m or n)
in	*X*	vector X
in	*incX*	increment for entries of X
in	*beta*	coefficient beta
in,out	*Y*	vector Y
in	*incY*	increment for entries of Y

Definition at line 83 of file amici_interface_cpp.cpp.

Here is the caller graph for this function:

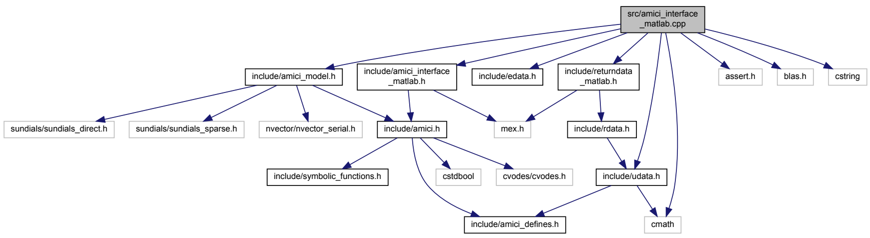


11.3 src/amici_interface_matlab.cpp File Reference

core routines for mex interface

```
#include "include/amici_interface_matlab.h"
#include "include/amici_model.h"
#include "include/edata.h"
#include "include/returndata_matlab.h"
#include "include/udata.h"
#include <assert.h>
#include <blas.h>
#include <cstring>
#include <cmath>
```

Include dependency graph for amici_interface_matlab.cpp:



Macros

- #define _USE_MATH_DEFINES
- #define M_PI 3.14159265358979323846
- #define readOptionScalar(OPTION, TYPE)
- #define readOptionData(OPTION)

Functions

- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])
- UserData ∗ userDataFromMatlabCall (const mxArray ∗prhs[ ], int nrhs, Model ∗model)

    userDataFromMatlabCall extracts information from the matlab call and returns the corresponding UserData struct
- char amici_blasCBlasTransToBlasTrans (AMICI_BLAS_TRANSPOSE trans)
- void amici_dgemm (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, AMICI_BLAS_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, const double beta, double ∗C, const int ldc)
- void amici_dgemv (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)
- ExpData ∗ expDataFromMatlabCall (const mxArray ∗prhs[ ], const UserData ∗udata, Model ∗model)

11.3.1 Detailed Description

This file defines the fuction mexFunction which is executed upon calling the mex file from matlab

11.3.2 Macro Definition Documentation

11.3.2.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

MS definition of PI and other constants

Definition at line 20 of file amici_interface_matlab.cpp.

11.3.2.2 M_PI

```
#define M_PI 3.14159265358979323846
```

define PI if we still have no definition

Definition at line 24 of file amici_interface_matlab.cpp.

11.3.2.3 readOptionScalar

```
#define readOptionScalar(
             OPTION,
             TYPE )
```

Value:

```
if (mxGetProperty(prhs[3], 0, #OPTION)) {                                    \
        udata->OPTION = (TYPE)mxGetScalar(mxGetProperty(prhs[3], 0, #OPTION)); \
    } else {                                                                   \
        warnMsgIdAndTxt("AMICI:mex:OPTION",                                    \
                        "Provided options do not have field " #OPTION "!");    \
        goto freturn;                                                          \
    }
```

@ brief extract information from a property of a matlab class (scalar) @ param OPTION name of the property @ param TYPE class to which the information should be cast

Definition at line 32 of file amici_interface_matlab.cpp.

11.3.2.4 readOptionData

```
#define readOptionData(
             OPTION )
```

Value:

```
if (mxGetProperty(prhs[3], 0, #OPTION)) {                                    \
        mxArray *a = mxGetProperty(prhs[3], 0, #OPTION);                       \
        int len = (int)mxGetM(a) * mxGetN(a);                                  \
        udata->OPTION = new double[len];                                       \
        memcpy(udata->OPTION, mxGetData(a), sizeof(double) * len);             \
    } else {                                                                   \
        warnMsgIdAndTxt("AMICI:mex:OPTION",                                    \
                        "Provided options do not have field " #OPTION "!");    \
        goto freturn;                                                          \
    }
```

@ brief extract information from a property of a matlab class (matrix) @ param OPTION name of the property

Definition at line 45 of file amici_interface_matlab.cpp.

11.3.3 Function Documentation

11.3.3.1 mexFunction()

```
void mexFunction (
            int nlhs,
            mxArray * plhs[],
            int nrhs,
            const mxArray * prhs[] )
```

mexFunction is the main interface function for the MATLAB interface. It reads in input data (udata and edata) and creates output data compound (rdata) and then calls the AMICI simulation routine to carry out numerical integration.

Parameters

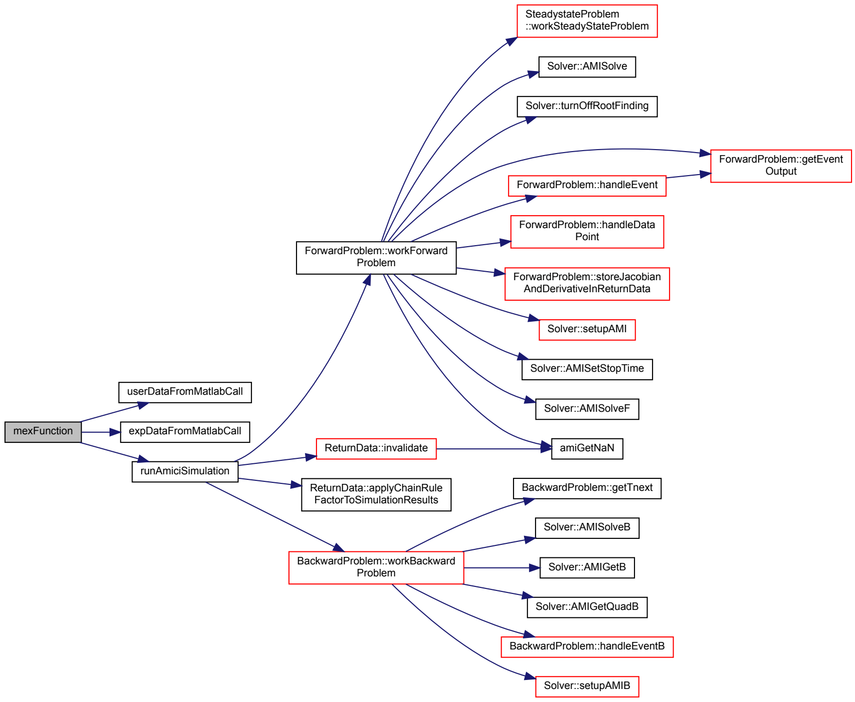
in	*nlhs*	number of output arguments of the matlab call **Type**: int
out	*plhs*	pointer to the array of output arguments **Type**: mxArray
in	*nrhs*	number of input arguments of the matlab call **Type**: int
in	*prhs*	pointer to the array of input arguments **Type**: mxArray

Returns

void

Definition at line 67 of file amici_interface_matlab.cpp.

Here is the call graph for this function:



11.3.3.2 userDataFromMatlabCall()

```
UserData* userDataFromMatlabCall (
            const mxArray * prhs[ ],
            int nrhs,
            Model * model )
```

userDataFromMatlabCall parses the input from the matlab call and writes it to an UserData class object

Parameters

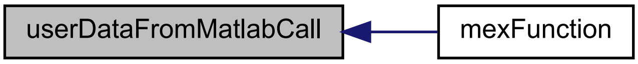
in	*nrhs*	number of input arguments of the matlab call **Type**: int
in	*prhs*	pointer to the array of input arguments **Type**: mxArray
in	*model*	pointer to the model object, this is necessary to perform dimension checks **Type**: Model

Returns

udata pointer to user data object
Type: UserData

Definition at line 118 of file amici_interface_matlab.cpp.

Here is the caller graph for this function:



11.3.3.3 amici_blasCBlasTransToBlasTrans()

```
char amici_blasCBlasTransToBlasTrans (
            AMICI_BLAS_TRANSPOSE trans )
```

amici_blasCBlasTransToBlasTrans translates AMICI_BLAS_TRANSPOSE values to CBlas readable strings

Parameters

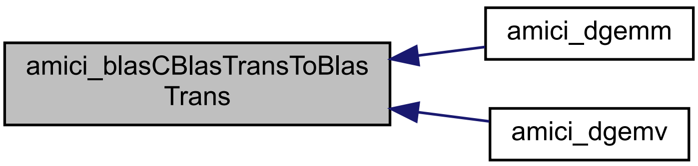
in	*trans*	flag indicating transposition and complex conjugation **Type**: AMICI_BLAS_TRANSPOSE

Returns

cblastrans CBlas readable CHAR indicating transposition and complex conjugation
Type: char

Definition at line 325 of file amici_interface_matlab.cpp.

Here is the caller graph for this function:



11.3.3.4 amici_dgemm()

```
void amici_dgemm (
            AMICI_BLAS_LAYOUT layout,
            AMICI_BLAS_TRANSPOSE TransA,
            AMICI_BLAS_TRANSPOSE TransB,
            const int M,
            const int N,
            const int K,
            const double alpha,
            const double * A,
            const int lda,
            const double * B,
            const int ldb,
            const double beta,
            double * C,
            const int ldc )
```

amici_dgemm provides an interface to the CBlas matrix matrix multiplication routine dgemm. This routines computes C = alpha*A*B + beta*C with A: [MxK] B:[KxN] C:[MxN]

Parameters

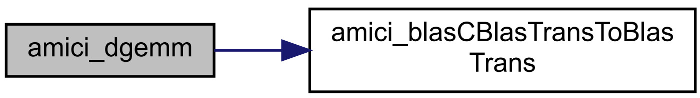
in	*layout*	always needs to be AMICI_BLAS_ColMajor.
in	*TransA*	flag indicating whether A should be transposed before multiplication
in	*TransB*	flag indicating whether B should be transposed before multiplication
in	*M*	number of rows in A/C
in	*N*	number of columns in B/C
in	*K*	number of rows in B, number of columns in A
in	*alpha*	coefficient alpha
in	*A*	matrix A
in	*lda*	leading dimension of A (m or k)
in	*B*	matrix B
in	*ldb*	leading dimension of B (k or n)
in	*beta*	coefficient beta
in,out	*C*	matrix C
in	*ldc*	leading dimension of C (m or n)

Returns

void

Definition at line 357 of file amici_interface_matlab.cpp.

Here is the call graph for this function:



11.3.3.5 amici_dgemv()

```
void amici_dgemv (
            AMICI_BLAS_LAYOUT layout,
            AMICI_BLAS_TRANSPOSE TransA,
            const int M,
            const int N,
            const double alpha,
            const double * A,
            const int lda,
            const double * X,
            const int incX,
            const double beta,
            double * Y,
            const int incY )
```

amici_dgemm provides an interface to the CBlas matrix vector multiplication routine dgemv. This routines computes y = alpha*A*x + beta*y with A: [MxN] x:[Nx1] y:[Mx1]

Parameters

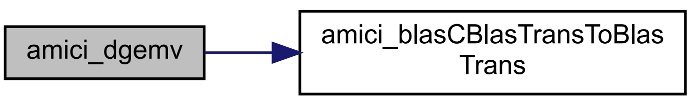
in	*layout*	always needs to be AMICI_BLAS_ColMajor.
in	*TransA*	flag indicating whether A should be transposed before multiplication
in	*M*	number of rows in A
in	*N*	number of columns in A
in	*alpha*	coefficient alpha
in	*A*	matrix A
in	*lda*	leading dimension of A (m or n)
in	*X*	vector X
in	*incX*	increment for entries of X
in	*beta*	coefficient beta
in,out	*Y*	vector Y
in	*incY*	increment for entries of Y

Returns

void

Definition at line 400 of file amici_interface_matlab.cpp.

Here is the call graph for this function:



11.3.3.6 expDataFromMatlabCall()

```
ExpData* expDataFromMatlabCall (
            const mxArray * prhs[ ],
            const UserData * udata,
            Model * model )
```

expDataFromMatlabCall parses the experimental data from the matlab call and writes it to an ExpData class object

Parameters

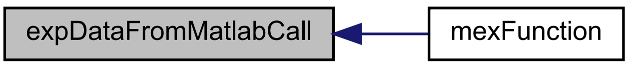
in	*prhs*	pointer to the array of input arguments **Type**: mxArray
in	*udata*	pointer to user data object **Type**: UserData
in	*model*	pointer to the model object, this is necessary to perform dimension checks **Type**: Model

Returns

edata pointer to experimental data object
Type: ExpData

Definition at line 428 of file amici_interface_matlab.cpp.

Here is the caller graph for this function:



11.4 src/spline.cpp File Reference

definition of spline functions

Functions

- int spline (int n, int end1, int end2, double slope1, double slope2, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])
- double seval (int n, double u, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])
- double sinteg (int n, double u, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])

11.4.1 Detailed Description

Author

Peter & Nigel, Design Software, 42 Gubberley St, Kenmore, 4069, Australia.

11.4.2 Function Documentation

11.4.2.1 spline()

```
int spline (
        int n,
        int end1,
        int end2,
        double slope1,
        double slope2,
        double x[ ],
        double y[ ],
        double b[ ],
        double c[ ],
        double d[ ] )
```

Evaluate the coefficients b[i], c[i], d[i], i = 0, 1, .. n-1 for a cubic interpolating spline

S(xx) = Y[i] + b[i] ∗ w + c[i] ∗ w∗∗2 + d[i] ∗ w∗∗3 where w = xx - x[i] and x[i] <= xx <= x[i+1]

The n supplied data points are x[i], y[i], i = 0 ... n-1.

Parameters

in	*n*	The number of data points or knots (n >= 2)
in	*end1*	0: default condition 1: specify the slopes at x[0]
in	*end2*	0: default condition 1: specify the slopes at x[n-1]
in	*slope1*	slope at x[0]
in	*slope2*	slope at x[n-1]
in	*x[ ]*	the abscissas of the knots in strictly increasing order
in	*y[ ]*	the ordinates of the knots
out	*b[ ]*	array of spline coefficients
out	*c[ ]*	array of spline coefficients
out	*d[ ]*	array of spline coefficients

Return values

0	normal return
1	less than two data points; cannot interpolate
2	x[] are not in ascending order

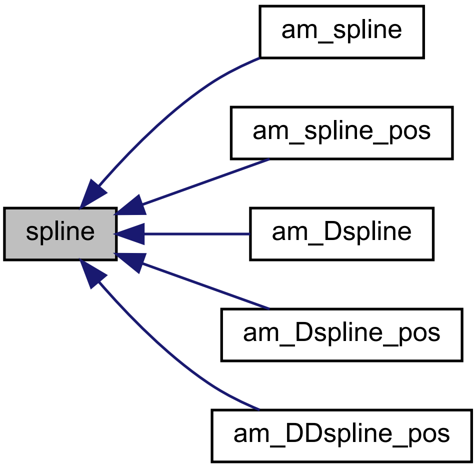
Notes

- The accompanying function seval() may be used to evaluate the spline while deriv will provide the first derivative.
- Using p to denote differentiation y[i] = S(X[i]) b[i] = Sp(X[i]) c[i] = Spp(X[i])/2 d[i] = Sppp(X[i])/6 (Derivative from the right)
- Since the zero elements of the arrays ARE NOW used here, all arrays to be passed from the main program should be dimensioned at least [n]. These routines will use elements [0 .. n-1].

- Adapted from the text Forsythe, G.E., Malcolm, M.A. and Moler, C.B. (1977) "Computer Methods for Mathematical Computations" Prentice Hall
- Note that although there are only n-1 polynomial segments, n elements are requird in b, c, d. The elements b[n-1], c[n-1] and d[n-1] are set to continue the last segment past x[n-1].

Definition at line 65 of file spline.cpp.

Here is the caller graph for this function:



11.4.2.2 seval()

```
double seval (
            int n,
            double u,
            double x[ ],
            double y[ ],
            double b[ ],
            double c[ ],
            double d[ ] )
```

Evaluate the cubic spline function

S(xx) = y[i] + b[i] ∗ w + c[i] ∗ w∗∗2 + d[i] ∗ w∗∗3 where w = u - x[i] and x[i] <= u <= x[i+1] Note that Horner's rule is used. If u < x[0] then i = 0 is used. If u > x[n-1] then i = n-1 is used.

Parameters

in	*n*	The number of data points or knots (n >= 2)
in	*u*	the abscissa at which the spline is to be evaluated
in	*x[ ]*	the abscissas of the knots in strictly increasing order
in	*y[ ]*	the ordinates of the knots
in	*b*	array of spline coefficients computed by spline().
in	*c*	array of spline coefficients computed by spline().
in	*d*	array of spline coefficients computed by spline().

Returns

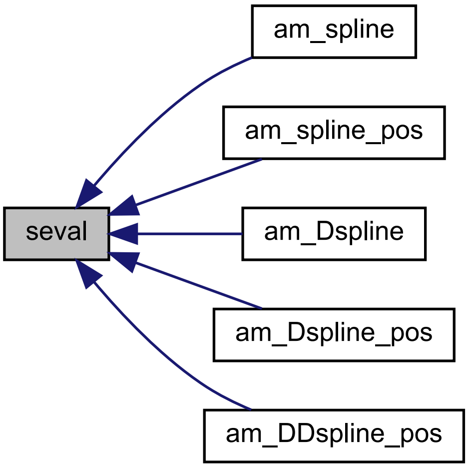
the value of the spline function at u

Notes

- If u is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

Definition at line 197 of file spline.cpp.

Here is the caller graph for this function:



11.4.2.3 sinteg()

```
double sinteg (
            int n,
            double u,
            double x[ ],
            double y[ ],
            double b[ ],
            double c[ ],
            double d[ ] )
```

Integrate the cubic spline function

S(xx) = y[i] + b[i] ∗ w + c[i] ∗ w∗∗2 + d[i] ∗ w∗∗3 where w = u - x[i] and x[i] <= u <= x[i+1]

The integral is zero at u = x[0].

If u < x[0] then i = 0 segment is extrapolated. If u > x[n-1] then i = n-1 segment is extrapolated.

Parameters

in	*n*	the number of data points or knots (n >= 2)
in	*u*	the abscissa at which the spline is to be evaluated
in	*x[ ]*	the abscissas of the knots in strictly increasing order
in	*y[ ]*	the ordinates of the knots
in	*b*	array of spline coefficients computed by spline().
in	*c*	array of spline coefficients computed by spline().
in	*d*	array of spline coefficients computed by spline().

Returns

the value of the spline function at u

Notes

- If u is not in the same interval as the previous call then a binary search is performed to determine the proper interval.

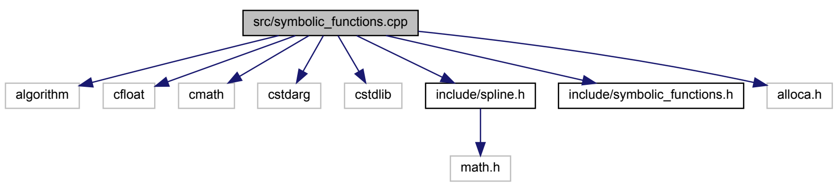
Definition at line 258 of file spline.cpp.

11.5 src/symbolic_functions.cpp File Reference

definition of symbolic functions

```
#include <algorithm>
#include <cfloat>
#include <cmath>
#include <cstdarg>
#include <cstdlib>
#include <include/spline.h>
#include <include/symbolic_functions.h>
#include <alloca.h>
```

Include dependency graph for symbolic_functions.cpp:



Functions

- int amiIsNaN (double what)
- int amiIsInf (double what)
- double amiGetNaN ()
- double amilog (double x)
- double dirac (double x)
- double heaviside (double x)
- double sign (double x)
- double am_min (double a, double b, double c)
- double Dam_min (int id, double a, double b, double c)
- double am_max (double a, double b, double c)
- double Dam_max (int id, double a, double b, double c)
- double am_spline (double t, int num,...)
- double am_spline_pos (double t, int num,...)
- double am_Dspline (int id, double t, int num,...)
- double am_Dspline_pos (int id, double t, int num,...)
- double am_DDspline (int id1, int id2, double t, int num,...)
- double am_DDspline_pos (int id1, int id2, double t, int num,...)

11.5.1 Detailed Description

This file contains definitions of various symbolic functions which

11.5.2 Function Documentation

11.5.2.1 amiIsNaN()

```
int amiIsNaN (
            double what )
```

c++ interface to the isNaN function

Parameters

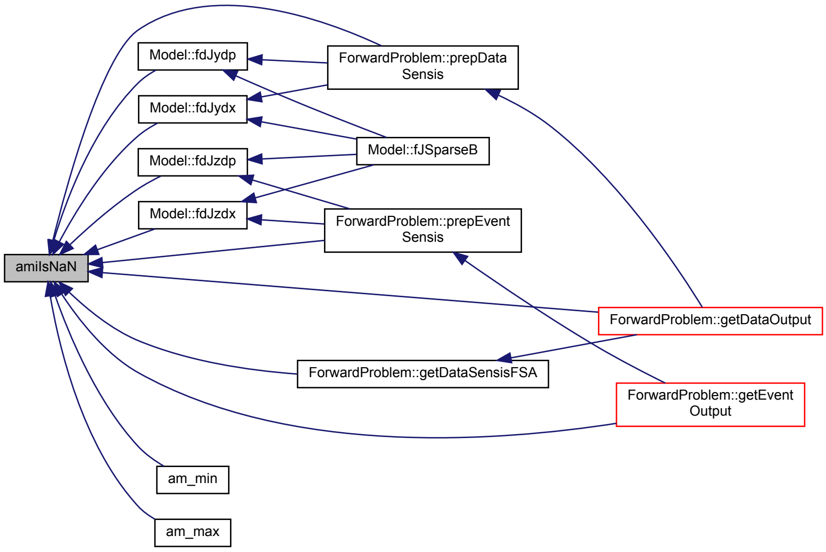
what	argument

Returns

isnan(what)

Definition at line 32 of file symbolic_functions.cpp.

Here is the caller graph for this function:



11.5.2.2 amiIsInf()

```
int amiIsInf (
            double what )
```

c++ interface to the isinf function

Parameters

what	argument

Returns

isnan(what)

Definition at line 43 of file symbolic_functions.cpp.

11.5.2.3 amiGetNaN()

```
double amiGetNaN ( )
```

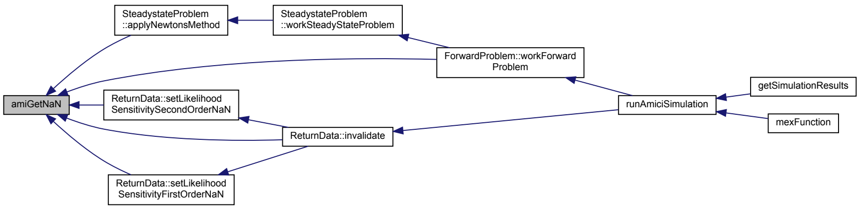
function returning nan

Returns

NaN

Definition at line 53 of file symbolic_functions.cpp.

Here is the caller graph for this function:



11.5.2.4 amilog()

```
double amilog (
            double x )
```

c implementation of log function, this prevents returning NaN values for negative values

Parameters

x	argument

Returns

if(x>0) then log(x) else -Inf

Definition at line 65 of file symbolic_functions.cpp.

11.5.2.5 dirac()

```
double dirac (
            double x )
```

c implementation of matlab function dirac

Parameters

x	argument

Returns

if(x==0) then INF else 0

Definition at line 80 of file symbolic_functions.cpp.

11.5.2.6 heaviside()

```
double heaviside (
            double x )
```

c implementation of matlab function heaviside

Parameters

x	argument

Returns

if(x>0) then 1 else 0

Definition at line 95 of file symbolic_functions.cpp.

11.5.2.7 sign()

```
double sign (
            double x )
```

c implementation of matlab function sign

Parameters

x	argument

Returns

0
Type: double

Definition at line 110 of file symbolic_functions.cpp.

11.5.2.8 am_min()

```
double am_min (
            double a,
            double b,
            double c )
```

c implementation of matlab function min

Parameters

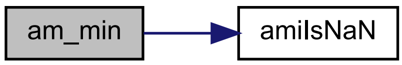
a	value1 **Type**: double
b	value2 **Type**: double
c	bogus parameter to ensure correct parsing as a function **Type**: double

Returns

if(a < b) then a else b
Type: double

Definition at line 131 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.9 Dam_min()

```
double Dam_min (
            int id,
            double a,
            double b,
            double c )
```

parameter derivative of c implementation of matlab function min

Parameters

id	argument index for differentiation
a	value1 **Type**: double
b	value2 **Type**: double
c	bogus parameter to ensure correct parsing as a function **Type**: double

Returns

id == 1: if(a < b) then 1 else 0
Type: double
id == 2: if(a < b) then 0 else 1
Type: double

Definition at line 154 of file symbolic_functions.cpp.

11.5.2.10 am_max()

```
double am_max (
            double a,
            double b,
            double c )
```

c implementation of matlab function max

Parameters

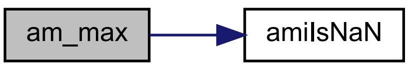
a	value1 **Type**: double
b	value2 **Type**: double
c	bogus parameter to ensure correct parsing as a function **Type**: double

Returns

if(a > b) then a else b
Type: double

Definition at line 179 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.11 Dam_max()

```
double Dam_max (
            int id,
            double a,
            double b,
            double c )
```

parameter derivative of c implementation of matlab function max

Parameters

id	argument index for differentiation
a	value1 **Type**: double
b	value2 **Type**: double
c	bogus parameter to ensure correct parsing as a function **Type**: double

Returns

id == 1: if(a $>$ b) then 1 else 0
Type: double
id == 2: if(a $>$ b) then 0 else 1
Type: double

Definition at line 202 of file symbolic_functions.cpp.

11.5.2.12 am_spline()

```
double am_spline (
            double t,
            int num,
             ... )
```

spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments must be of type double.

Parameters

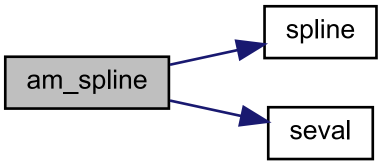
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

spline(t)

Definition at line 229 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.13 am_spline_pos()

```
double am_spline_pos (
            double t,
            int num,
             ... )
```

exponentiated spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments must be of type double.

Parameters

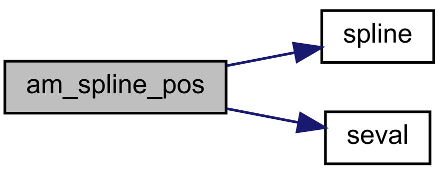
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

spline(t)

Definition at line 280 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.14 am_Dspline()

```
double am_Dspline (
            int id,
            double t,
            int num,
             ... )
```

derivation of a spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments but id must be of type double.

Parameters

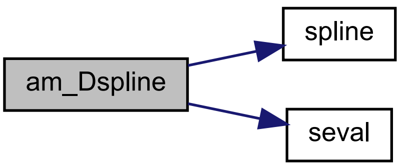
id	index of node to which the derivative of the corresponding spline coefficient should be computed
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

dsplinedp(t)

Definition at line 332 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.15 am_Dspline_pos()

```
double am_Dspline_pos (
            int id,
            double t,
            int num,
             ... )
```

derivation of an exponentiated spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments but id must be of type double.

Parameters

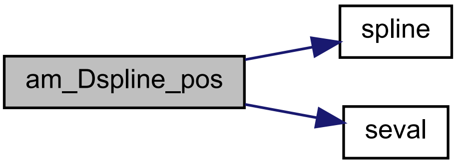
id	index of node to which the derivative of the corresponding spline coefficient should be computed
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

dsplinedp(t)

Definition at line 388 of file symbolic_functions.cpp.

Here is the call graph for this function:



11.5.2.16 am_DDspline()

```
double am_DDspline (
            int id1,
            int id2,
            double t,
            int num,
             ... )
```

second derivation of a spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments but id1 and id2 must be of type double.

Parameters

id1	index of node to which the first derivative of the corresponding spline coefficient should be computed
id2	index of node to which the second derivative of the corresponding spline coefficient should be computed
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

ddspline(t)

Definition at line 453 of file symbolic_functions.cpp.

11.5.2.17 am_DDspline_pos()

```
double am_DDspline_pos (
            int id1,
            int id2,
            double t,
            int num,
             ... )
```

derivation of an exponentiated spline function, takes variable argument pairs (ti,pi) with `ti`: location of node i and `pi`: spline value at node i. the last two arguments are always `ss`: flag indicating whether slope at first node should be user defined and `dudt` user defined slope at first node. All arguments but id1 and id2 must be of type double.

Parameters

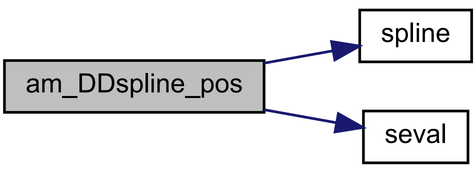
id1	index of node to which the first derivative of the corresponding spline coefficient should be computed
id2	index of node to which the second derivative of the corresponding spline coefficient should be computed
t	point at which the spline should be evaluated
num	number of spline nodes

Returns

ddspline(t)

Definition at line 468 of file symbolic_functions.cpp.

Here is the call graph for this function:



Index