

AMICI

Generated by Doxygen 1.8.14

Contents

1	About AMICI	1
2	License Conditions	2
3	How to contribute	2
4	Installation	3
5	Python Interface	6
6	MATLAB Interface	8
7	C++ Interface	13
8	FAQ	14
9	Namespace Documentation	14
9.1	amici Namespace Reference	14
9.2	amici.ode_export Namespace Reference	75
9.3	amici.plotting Namespace Reference	80
9.4	amici.sbml_import Namespace Reference	81
10	Class Documentation	85
10.1	AmiException Class Reference	85
10.2	AmiVector Class Reference	88
10.3	AmiVectorArray Class Reference	98
10.4	BackwardProblem Class Reference	104
10.5	CvodeException Class Reference	110
10.6	ExpData Class Reference	111
10.7	ForwardProblem Class Reference	138
10.8	IDAException Class Reference	146
10.9	IntegrationFailure Class Reference	147
10.10	IntegrationFailureB Class Reference	149
10.11	Model Class Reference	150

10.12Model_DAE Class Reference	281
10.13Model_ODE Class Reference	314
10.14NewtonFailure Class Reference	344
10.15NewtonSolver Class Reference	345
10.16NewtonSolverDense Class Reference	352
10.17NewtonSolverIterative Class Reference	355
10.18NewtonSolverSparse Class Reference	359
10.19Constant Class Reference	361
10.20Expression Class Reference	362
10.21LogLikelihood Class Reference	364
10.22ModelQuantity Class Reference	365
10.23Observable Class Reference	368
10.24ODEExporter Class Reference	369
10.25ODEModel Class Reference	373
10.26Parameter Class Reference	386
10.27SigmaY Class Reference	387
10.28State Class Reference	388
10.29TemplateAmici Class Reference	390
10.30ReturnData Class Reference	391
10.31SBMLEException Class Reference	409
10.32SbmlImporter Class Reference	409
10.33SetupFailure Class Reference	423
10.34Solver Class Reference	424
10.35SteadystateProblem Class Reference	493
10.36amidata Class Reference	500
10.37amievent Class Reference	505
10.38amifun Class Reference	508
10.39amimodel Class Reference	515
10.40amioption Class Reference	539
10.41amised Class Reference	547
10.42optsym Class Reference	550

11 File Documentation	551
11.1 am_and.m File Reference	551
11.2 am_eq.m File Reference	553
11.3 am_ge.m File Reference	554
11.4 am_gt.m File Reference	554
11.5 am_if.m File Reference	555
11.6 am_le.m File Reference	556
11.7 am_lt.m File Reference	557
11.8 am_max.m File Reference	558
11.9 am_min.m File Reference	559
11.10am_or.m File Reference	560
11.11am_piecewise.m File Reference	561
11.12am_stepfun.m File Reference	562
11.13am_xor.m File Reference	563
11.14amici.cpp File Reference	564
11.15amiwrap.m File Reference	564
11.16cblas.cpp File Reference	565
11.17interface_matlab.cpp File Reference	566
11.18SBML2AMICI.m File Reference	568
11.19spline.cpp File Reference	569
11.20symbolic_functions.cpp File Reference	569
Index	571

1 About AMICI

AMICI provides a multilanguage (Python, C++, Matlab) interface for the SUNDIALS solvers CVODES (for ordinary differential equations) and IDAS (for algebraic differential equations). AMICI allows the user to read differential equation models specified as SBML and automatically compiles such models as .mex simulation files, C++ executables or python modules. In contrast to the SUNDIALSTB interface, all necessary functions are transformed into native C++ code, which allows for a significantly faster simulation. Beyond forward integration, the compiled simulation file also allows for 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.

Online documentation is available as [github-pages](#).

Publications

Fröhlich, F., Kaltenbacher, B., Theis, F. J., & Hasenauer, J. (2017). Scalable Parameter Estimation for Genome-Scale Biochemical Reaction Networks. *Plos Computational Biology*, 13(1), e1005331. doi: 10.1371/journal.pcbi.1005331

Fröhlich, F., Theis, F. J., Rädler, J. O., & Hasenauer, J. (2017). Parameter estimation for dynamical systems with discrete events and logical operations. *Bioinformatics*, 33(7), 1049–1056. doi: 10.1093/bioinformatics/btw764

Current build status

2 License Conditions

Copyright (c) 2015-2018, Fabian Fröhlich, Jan Hasenauer, Daniel Weindl and Paul Stapor All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

3 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/buildAll.sh && scripts/run-cpputest.sh`
 - Run `tests/testModels.m`
- When all tests are passing and you think your code is ready to merge, request a code review

Adding/Updating tests

To add new tests add a new corresponding python script (see, e.g., `tests/example_dirac.py`) and add it to and run `tests/generateTestConfigurationForExamples.sh`. To update test results replace `tests/cpputest/expectedResults.h5` by `tests/cpputest/writeResults.h5.bak` [ONLY DO THIS AFTER TRIPLE CHECKING CORRECTNESS OF RESULTS]. Before replacing the test results, confirm that only expected datasets have changed, e.g. using `h5diff -v -r 1e-8 tests/cpputest/expectedResults.h5 tests/cpputest/writeResults.h5.bak | less`

4 Installation

Availability

The sources for AMICI are accessible as

- Source [tarball](#)
- Source [zip](#)
- GIT repository on [github](#)

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/ICB-DCM/AMICI> and a direct clone is possible via

```
git clone https://github.com/ICB-DCM/AMICI.git AMICI
```

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.

Dependencies

The MATLAB interface only depends on the symbolic toolbox, which is needed for model compilation, but not simulation.

Symbolic Engine

The MATLAB interface requires the symbolic toolbox for model compilation. The symbolic toolbox is not required for model simulation.

Math Kernel Library (MKL)

The python and C++ interfaces require a system installation of BLAS. AMICI has been tested with various native and general purpose MKL implementations such as Accelerate, Intel MKL, cblas, openblas, atlas. The matlab interface uses the MATLAB MKL, which requires no prior installation.

HDF5

The python and C++ interfaces provide routines to read and write options and results in hdf5 format. For the python interface, the installation of hdf5 is optional, but for the C++ interface it is required. HDF can be installed using package managers such as [brew](#) or [apt](#):

```
brew install hdf5
```

or

```
apt-get install libhdf5-serial-dev
```

SWIG

The python interface requires SWIG, which has to be installed by the user. Swig can be installed using package managers such as [brew](#) or [apt](#):

```
brew install swig
```

or

```
apt-get install swig3.0
```

We note here that some linux package managers may provide swig executables as `swig3.0`, but installation as `swig` is required. This can be fixed using, e.g., symbolic links:

```
mkdir -p ~/bin/ && ln -s $(which swig3.0) ~/bin/swig && export PATH=~/bin/:$PATH
```

python packages

The python interface requires the python packages `pkgconfig` and `numpy` to be installed before AMICI can be installed. These can be installed via `pip`:

```
pip3 install pkgconfig numpy
```

MATLAB

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

```
installAMICI.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.com](#) Note that Microsoft Visual Studio compilers are currently not supported.

python

To use AMICI from python, install the module and all other requirements using pip

```
pip3 install amici
```

You can now import it as python module:

```
import amici
```

C++

To use AMICI from C++, run the

```
./scripts/buildSundials.sh  
./scripts/buildSuiteSparse.sh  
./scripts/buildAmici.sh
```

script to compile amici library. The static library file can then be linked from

```
./build/libamici.a
```

In CMake-based packages, amici can be linked via

```
find_package(Amici)
```

Dependencies

The MATLAB interface requires the Mathworks Symbolic Toolbox for model generation via `amiwrap(...)`, but not for execution of precompiled models. Currently MATLAB R2018a or newer is not supported (see <https://github.com/ICB-DCM/AMICI/issues/307>)

The python interface requires python 3.6 or newer and `cblas` library to be installed. Windows installations via pip are currently not supported, but users may try to install amici using the build scripts provided for the C++ interface (these will by default automatically install the python module).

The C++ interface requires `cmake` and `cblas` to be installed.

The tools SUNDIALS and SuiteSparse shipped with AMICI do **not** require explicit 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](#)

5 Python Interface

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

Model Definition

This guide will guide the user on how to specify models in Python using SBML. For example implementations see the examples in the python/examples directory.

SBML input

First, import an sbml file using the `amici.sbml_importer.SbmlImporter` class:

```
import amici
sbmlImporter = amici.SbmlImporter('model_steadystate_scaled.sbml')
```

the sbml document as imported by `libSBML` is available as

```
sbml = sbmlImporter.sbml
```

Constants

parameters that should be considered constants can be specified in a list of strings specifying the respective SbmId of a parameter.

```
constantParameters=['k4']
```

Observables

assignment rules that should be considered as observables can be extracted using the `amici.assignmentRules2observables` function

```
observables = amici.assignmentRules2observables(sbml, filter=lambda variableId:  
                                                variableId.startswith('observable_') and not variableId.endswith('_'))
```

Standard Deviations

standard deviations can be specified as dictionaries ...

```
sigmas = {'observable_xlwithsigma': 'observable_xlwithsigma_sigma'}
```

Model Compilation

to compile the sbml as python module, the user has to call the method `amici.sbml_import.SbmlImporter.sbml2amici`, passing all the previously defined model specifications

```
sbmlImporter.sbml2amici('test', 'test',  
                        observables=observables,  
                        constantParameters=constantParameters,  
                        sigmas=sigma)
```

Model Simulation

currently the model folder has to be manually added to the python path

```
import sys  
sys.path.insert(0, 'test')
```

the compiled model can now be imported as python module

```
import test as modelModule
```

to obtain a model instance call the `getModel()` method. This model instance will be instantiated using the default parameter values specified in the sbml.

```
model = modelModule.getModel()
```

then pass the simulation timepoints to `amici.Model.setTimepoints`

```
model.setTimepoints(np.linspace(0, 60, 60))
```

for simulation we need to generate a solver instance

```
solver = model.getSolver()
```

the model simulation can now be carried out using `amici.runAmiciSimulation`

```
rdata = amici.runAmiciSimulation(model, solver)
```

6 MATLAB Interface

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

Model Definition

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

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()
```

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	default 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 false, the fields 'forward' and 'adjoint' will speed up the time required to compile the model but also disable the respective sensitivity computation.

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:

```
model.sym.x = [ state1 state2 state3 ];
```

Parameters

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

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

Create the parameters vector

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

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

```
model.sym.k = [ const1 const2 ];
```

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

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

or

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

The specification of f or xdot may depend on states, parameters and constants.

For DAEs also specify the mass matrix.

```
model.sym.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 [src/symbolic_functions.cpp](#).

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 unknown time/state dependence.

Initial Conditions

Specify the initial conditions. These may depend on parameters or constants and must have the same size as x.

```
model.sym.x0 = [ param4, 0, 0 ];
```

Observables

Specify the observables. These may depend on parameters and constants.

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

In the definition of the observable you can use certain symbolic functions. For a full list of available functions see [src/symbolic_functions.cpp](#). Dirac functions in observables will have no effect.

Events

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

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

Events may depend on states, parameters and constants but **not** on observables.

For more details about event support see <https://doi.org/10.1093/bioinformatics/btw764>

Standard Deviation

Specifying 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 deviation for observable data is denoted by `sigma_y`

```
model.sym.sigma_y(1) = param5;
```

Standard deviation for event data is denoted by `sigma_t`

```
model.sym.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 `D.Sigma_Y` or `D.Sigma_T` in the user-provided data struct is NaN. See `simulation` for details.

Objective Function

By default, AMICI assumes a normal noise model and uses the corresponding negative log-likelihood

```
J = 1/2 * sum(((y_i(t)-my_t)/sigma_y_i)^2 + log(2*pi*sigma_y^2)
```

as objective function. A user provided objective function can be specified in

```
model.sym.Jy
```

As reference see the default specification of `this.sym.Jy` in [amimodel.makeSyms](#).

SBML

AMICI can also import SBML models using the command `SBML2AMICI`. This will generate a model specification as described above, which may be edited by the user to apply further changes.

Model Compilation

The model can then be compiled by calling `amiwrap.m`:

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

Here `modelname` should be a string defining the name of the model, `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 models and pass them directly to `amiwrap(...)` without generating respective model definition scripts.

Model Simulation

After the call to `amiwrap(...)` two files will be placed in the specified directory. One is a `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.

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 event outputs will then be available as `sol.z`. If no event occurred there will be an event at the end of the considered interval with the final value of the root function stored in `sol.rz`.

Alternatively the integration can also be called via

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

The integration status will be indicated by the flag `status`. 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.

Forward Sensitivities

Set the sensitivity computation to forward sensitivities and integrate:

```
options.sensi = 1;
options.sensi_meth = 'forward';
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `sol.status` flag. Negative values indicate failed integration. The states will be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will be available as `sol.y`, with the derivative with respect to the parameters in `sol.sy`. The event outputs will be available as `sol.z`, with the derivative with respect to the parameters in `sol.sz`. If no event occurred there will be an event at the end of the considered interval with the final value of the root function stored in `sol.rz`, with the derivative with respect to the parameters in `sol.srz`.

Alternatively the integration can 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 indicate 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.

Adjoint Sensitivities

Set the sensitivity computation to adjoint sensitivities:

```
options.sensi = 1;
options.sensi_meth = 'adjoint';
```

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 `model.sym.sigma_y` and `model.sym.sigma_t`. Data points with `NaN` value will be completely ignored.

Integrate:

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

The integration status will be indicated by the `sol.status` flag. Negative values indicate 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.

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.

Set the final timepoint as infinity, this will indicate the solver to compute the steady state:

```
t = Inf;
```

Set the sensitivity computation to steady state sensitivities:

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

Integrate:

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

The states will be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will 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.diagnosis.xdot`.

7 C++ Interface

The [Python Interface](#) and [MATLAB Interface](#) can translate the model definition into C++ code, which is then compiled into a .mex file or a python module. Advanced users can also use this code within stand-alone C/C++ application for use in other 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` and `amici.SbmlImporter.sbml2amici` 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
[... many more files model_steadystate_*.cpp|h|md5|o]
wrapfunctions.cpp
wrapfunctions.h
model_steadystate.h
```

Running a simulation

The entry function for running an AMICI simulation is `runAmiciSimulation(...)`, declared in `amici.h`. This function requires (i) a `Model` instance. For the example `model_steadystate` the respective class is provided as `Model_model_steadystate` in `model_steadystate.h`. For convenience, the header `wrapfunctions.h` defines a function `getModel()`, that returns an instance of that class. (ii) a `Solver` instance. This solver instance needs to match the requirements of the model and can be generated using `model->getSolver()`. (iii) optionally an `ExpData` instance, which contains any experimental data.

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. `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/*.cpp`, the SUNDIALS and the SUITESPARSE library, or use the CMake package configuration from the build directory which tells CMake about all AMICI dependencies.

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`.

8 FAQ

Q: My model fails to build.

A: Remove the corresponding model directory located in `AMICI/models/*yourmodelName*` and compile again.

Q: It still does not compile.

A: Make an `issue` and we will have a look.

Q: I get an out of memory error while compiling my model on a Windows machine.

A: This may be due to an old compiler version. See [issue #161](#) for instructions on how to install a new compiler.

Q: The simulation/sensitivities I get are incorrect.

A: There are some known issues, especially with adjoint sensitivities, events and DAEs. If your particular problem is not featured in the `issues` list, please add it!

9 Namespace Documentation

9.1 amici Namespace Reference

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Namespaces

- [ode_export](#)
The C++ ODE export module for python.
- [plotting](#)
Plotting related functions.
- [sbml_import](#)
The python sbml import module for python.

Classes

- class [AmiException](#)
amici exception handler class
- class [AmiVector](#)
- class [AmiVectorArray](#)
- class [BackwardProblem](#)
class to solve backwards problems.
- class [CvodeException](#)
cicode exception handler class
- class [ExpData](#)
ExpData carries all information about experimental or condition-specific data.
- class [ForwardProblem](#)
The ForwardProblem class groups all functions for solving the backwards problem. Has only static members.
- class [IDAException](#)
ida exception handler class
- class [IntegrationFailure](#)
integration failure exception for the forward problem this exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user
- class [IntegrationFailureB](#)
integration failure exception for the backward problem this exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user
- class [Model](#)
The Model class represents an AMICI ODE model. The model can compute various model related quantities based on symbolically generated code.
- class [Model_DAE](#)
The Model class represents an AMICI DAE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.
- class [Model_ODE](#)
The Model class represents an AMICI ODE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.
- class [NewtonFailure](#)
newton failure exception this exception should be thrown when the steady state computation failed to converge for this exception we can assume that we can recover from the exception and return a solution struct to the user
- class [NewtonSolver](#)
The NewtonSolver class sets up the linear solver for the Newton method.
- class [NewtonSolverDense](#)
The NewtonSolverDense provides access to the dense linear solver for the Newton method.
- class [NewtonSolverIterative](#)
The NewtonSolverIterative provides access to the iterative linear solver for the Newton method.
- class [NewtonSolverSparse](#)

The [NewtonSolverSparse](#) provides access to the sparse linear solver for the Newton method.

- class [ReturnData](#)
class that stores all data which is later returned by the mex function
- class [SetupFailure](#)
setup failure exception this exception should be thrown when the solver setup failed for this exception we can assume that we cannot recover from the exception and an error will be thrown
- class [Solver](#)
- class [SteadystateProblem](#)
The [SteadystateProblem](#) class solves a steady-state problem using Newton's method and falls back to integration on failure.

TypeDefs

- typedef double [realtype](#)
- typedef void(* [msgIdAndTxtFp](#)) (const char *identifier, const char *format,...)
msgIdAndTxtFp

Enumerations

- enum [BLASLayout](#) { **rowMajor** = 101, **colMajor** = 102 }
- enum [BLASTranspose](#) { **noTrans** = 111, **trans** = 112, **conjTrans** = 113 }
- enum [ParameterScaling](#) { **none**, **In**, **log10** }
- enum [SecondOrderMode](#) { **none**, **full**, **directional** }
- enum [SensitivityOrder](#) { **none**, **first**, **second** }
- enum [SensitivityMethod](#) { **none**, **forward**, **adjoint** }
- enum [LinearSolver](#) {
dense = 1, **band** = 2, **LAPACKDense** = 3, **LAPACKBand** = 4,
diag = 5, **SPGMR** = 6, **SPBCG** = 7, **SPTFQMR** = 8,
KLU = 9 }
- enum [InternalSensitivityMethod](#) { **simultaneous** = 1, **staggered** = 2, **staggered1** = 3 }
- enum [InterpolationType](#) { **hermite** = 1, **polynomial** = 2 }
- enum [LinearMultistepMethod](#) { **adams** = 1, **BDF** = 2 }
- enum [NonlinearSolverIteration](#) { **functional** = 1, **newton** = 2 }
- enum [StateOrdering](#) { **AMD**, **COLAMD**, **natural** }
- enum [SteadyStateSensitivityMode](#) { **newtonOnly**, **simulationFSA** }
- enum [NewtonStatus](#) { **failed** = -1, **newt** = 1, **newt_sim** = 2, **newt_sim_newt** = 3 }
- enum [mexRhsArguments](#) {
RHS_TIMEPOINTS, **RHS_PARAMETERS**, **RHS_CONSTANTS**, **RHS_OPTIONS**,
RHS_PLIST, **RHS_XSCALE_UNUSED**, **RHS_INITIALIZATION**, **RHS_DATA**,
RHS_NUMARGS_REQUIRED = **RHS_DATA**, **RHS_NUMARGS** }

The [mexFunctionArguments](#) enum takes care of the ordering of mex file arguments (indexing in prhs)

Functions

- void [printErrMsgIdAndTxt](#) (const char *identifier, const char *format,...)
- void [printWarnMsgIdAndTxt](#) (const char *identifier, const char *format,...)
- std::unique_ptr< [ReturnData](#) > [runAmiciSimulation](#) ([Solver](#) &solver, const [ExpData](#) *edata, [Model](#) &model)
- std::vector< std::unique_ptr< [ReturnData](#) > > [runAmiciSimulations](#) ([Solver](#) const &solver, const std::vector< [ExpData](#) *> &edatas, [Model](#) const &model, int num_threads)
- void [amici_dgemv](#) ([BLASLayout](#) layout, [BLASTranspose](#) 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)

- void `amici_dgemm` (`BLASLayout` layout, `BLASTranspose` TransA, `BLASTranspose` 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_daxpy` (int n, double alpha, const double *x, const int incx, double *y, int incy)

*Compute $y = a*x + y$.*
- void `setModelData` (const mxArray *prhs[], int nrhs, `Model` &model)

setModelData sets data from the matlab call to the model object
- void `setSolverOptions` (const mxArray *prhs[], int nrhs, `Solver` &solver)

setSolverOptions solver options from the matlab call to a solver object
- `ReturnDataMatlab` * `setupReturnData` (mxArray *plhs[], int nlhs)
- std::unique_ptr<`ExpData`> `expDataFromMatlabCall` (const mxArray *prhs[], const `Model` &model)
- int `checkFinite` (const int N, const `realtype` *array, const char *fun)
- void `unscaleParameters` (const double *bufferScaled, const `ParameterScaling` *pscale, int n, double *buffer ← `Unscaled`)

Remove parameter scaling according to the parameter scaling in pscale.
- void `unscaleParameters` (std::vector<double> &bufferScaled, std::vector<`ParameterScaling`> &pscale, std::vector<double> &bufferUnscaled)

Remove parameter scaling according to the parameter scaling in pscale.
- double `getUnscaledParameter` (double scaledParameter, `ParameterScaling` scaling)

Remove parameter scaling according to scaling
- bool `operator==` (const `Model` &a, const `Model` &b)
- mxArray * `getReturnDataMatlabFromAmiciCall` (`ReturnData` const *rdata)
- mxArray * `initMatlabReturnFields` (`ReturnData` const *rdata)
- mxArray * `initMatlabDiagnosisFields` (`ReturnData` const *rdata)
- template<typename T>
 void `writeMatlabField0` (mxArray *matlabStruct, const char *fieldName, T fieldData)
- template<typename T>
 void `writeMatlabField1` (mxArray *matlabStruct, const char *fieldName, std::vector<T> fieldData, const int dim0)
- template<typename T>
 void `writeMatlabField2` (mxArray *matlabStruct, const char *fieldName, std::vector<T> fieldData, int dim0, int dim1, std::vector<int> perm)
- template<typename T>
 void `writeMatlabField3` (mxArray *matlabStruct, const char *fieldName, std::vector<T> fieldData, int dim0, int dim1, int dim2, std::vector<int> perm)
- template<typename T>
 void `writeMatlabField4` (mxArray *matlabStruct, const char *fieldName, std::vector<T> fieldData, int dim0, int dim1, int dim2, int dim3, std::vector<int> perm)
- double * `initAndAttachArray` (mxArray *matlabStruct, const char *fieldName, std::vector<mwSize> dim)
- void `checkFieldNames` (const char **fieldNames, const int fieldCount)
- template<typename T>
 std::vector<T> `reorder` (const std::vector<T> input, const std::vector<int> order)
- template<typename T>
 char * `serializeToChar` (T const &data, int *size)
- template<typename T>
 T `deserializeFromChar` (const char *buffer, int size)
- template<typename T>
 std::string `serializeToString` (T const &data)
- template<typename T>
 std::vector<char> `serializeToStdVec` (T const &data)
- template<typename T>
 T `deserializeFromString` (std::string const &serialized)
- bool `operator==` (const `Solver` &a, const `Solver` &b)
- int `spline` (int n, int end1, int end2, double slope1, double slope2, double x[], double y[], double b[], double c[], double d[])

Compute a cubic spline approximation between (end1, slope1) and (end2, slope2).

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

Evaluate the cubic spline function.
- `double sinteg` (int n, double u, double x[], double y[], double b[], double c[], double d[])
- `double log` (double x)
- `double dirac` (double x)
- `double heaviside` (double x)
- `double min` (double a, double b, double c)
- `double Dmin` (int id, double a, double b, double c)
- `double max` (double a, double b, double c)
- `double Dmax` (int id, double a, double b, double c)
- `double pos_pow` (double base, double exponent)
- `int isNaN` (double what)
- `int isInf` (double what)
- `double getNaN()`
- `double sign` (double x)
- `double spline` (double t, int num,...)
- `double spline_pos` (double t, int num,...)
- `double Dspline` (int id, double t, int num,...)
- `double Dspline_pos` (int id, double t, int num,...)
- `double DDspline` (int id1, int id2, double t, int num,...)
- `double DDspline_pos` (int id1, int id2, double t, int num,...)
- `def runAmiciSimulation` (model, solver, edata=None)

Convenience wrapper around `amici.runAmiciSimulation` (generated by swig)
- `def ExpData` (args)

Convenience wrapper for `ExpData` constructors.
- `def runAmiciSimulations` (model, solver, edata_list, num_threads=1)

Convenience wrapper for loops of `amici.runAmiciSimulation`.
- `int dbl2int` (const double x)
- `char amici_blasCBlasTransToBlasTrans` (BLASTranspose trans)
- `std::vector< realtype > mxArrayToVector` (const mxArray *array, int length)
- `realtype getValueById` (std::vector< std::string > const &ids, std::vector< realtype > const &values, std::vector< std::string > const &id, const char *variable_name, const char *id_name)

local helper function to get parameters
- `void setValueById` (std::vector< std::string > const &ids, std::vector< realtype > &values, realtype value, std::vector< std::string > const &id, const char *variable_name, const char *id_name)

local helper function to set parameters
- `int setValueByIdRegex` (std::vector< std::string > const &ids, std::vector< realtype > &values, realtype value, std::string const ®ex, const char *variable_name, const char *id_name)

local helper function to set parameters via regex

Variables

- `msgIdAndTxtFp` errMsgIdAndTxt = &printErrMsgIdAndTxt
- `msgIdAndTxtFp` warnMsgIdAndTxt = &printWarnMsgIdAndTxt
- `constexpr double pi` = 3.14159265358979323846
- `bool hdf5_enabled` = False

boolean indicating if amici was compiled with hdf5 support
- `bool has_clibs` = False

boolean indicating if this is the full package with swig interface or the raw package without
- `amici_path`

absolute root path of the amici repository
- `amiciSwigPath` = os.path.join(amici_path, 'swig')

- *absolute path of the amici swig directory*
- `amiciSrcPath = os.path.join(amici_path, 'src')`
absolute path of the amici source directory
- `amiciModulePath = os.path.dirname(__file__)`
absolute root path of the amici module

9.1.1 Detailed Description

The AMICI Python module provides functionality for importing SBML models and turning them into C++ Python extensions.

Getting started:

```
creating a extension module for an SBML model:  

import amici  

amiSbml = amici.SbmlImporter('mymodel.sbml')  

amiSbml.sbml2amici('modelName', 'outputDirectory')  
  

using the created module (set python path)  

import modelName  

help(modelName)
```

9.1.2 Typedef Documentation

9.1.2.1 realtype

```
typedef double realtype  
  
defines variable type for simulation variables (determines numerical accuracy)
```

Definition at line 51 of file defines.h.

9.1.2.2 msgIdAndTxtFp

```
typedef void(* msgIdAndTxtFp) (const char *identifier, const char *format,...)
```

Parameters

<i>identifier</i>	string with error message identifier
<i>format</i>	string with error message printf-style format
...	arguments to be formatted

Definition at line 159 of file defines.h.

9.1.3 Enumeration Type Documentation

9.1.3.1 BLASLayout

enum **BLASLayout** [strong]

BLAS Matrix Layout, affects dgemm and gemv calls

Definition at line 54 of file defines.h.

9.1.3.2 BLASTranspose

enum **BLASTranspose** [strong]

BLAS Matrix Transposition, affects dgemm and gemv calls

Definition at line 60 of file defines.h.

9.1.3.3 ParameterScaling

enum **ParameterScaling** [strong]

modes for parameter transformations

Definition at line 67 of file defines.h.

9.1.3.4 SecondOrderMode

enum **SecondOrderMode** [strong]

modes for second order sensitivity analysis

Definition at line 74 of file defines.h.

9.1.3.5 SensitivityOrder

enum **SensitivityOrder** [strong]

orders of sensitivity analysis

Definition at line 81 of file defines.h.

9.1.3.6 SensitivityMethod

enum `SensitivityMethod` [strong]

methods for sensitivity computation

Definition at line 88 of file defines.h.

9.1.3.7 LinearSolver

enum `LinearSolver` [strong]

linear solvers for CVODES/IDAS

Definition at line 95 of file defines.h.

9.1.3.8 InternalSensitivityMethod

enum `InternalSensitivityMethod` [strong]

CVODES/IDAS forward sensitivity computation method

Definition at line 108 of file defines.h.

9.1.3.9 InterpolationType

enum `InterpolationType` [strong]

CVODES/IDAS state interpolation for adjoint sensitivity analysis

Definition at line 115 of file defines.h.

9.1.3.10 LinearMultistepMethod

enum `LinearMultistepMethod` [strong]

CVODES/IDAS linear multistep method

Definition at line 121 of file defines.h.

9.1.3.11 NonlinearSolverIteration

enum `NonlinearSolverIteration` [strong]

CVODES/IDAS Nonlinear Iteration method

Definition at line 127 of file defines.h.

9.1.3.12 StateOrdering

enum `StateOrdering` [strong]

KLU state reordering

Definition at line 133 of file defines.h.

9.1.3.13 SteadyStateSensitivityMode

enum `SteadyStateSensitivityMode` [strong]

Sensitivity computation mode in steadyStateProblem

Definition at line 140 of file defines.h.

9.1.3.14 NewtonStatus

enum `NewtonStatus` [strong]

State in which the steady state computation finished

Definition at line 146 of file defines.h.

9.1.4 Function Documentation

9.1.4.1 printErrMsgIdAndTxt()

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

`printErrMsgIdAndTxt` prints a specified error message associated to the specified identifier

Parameters

in	<i>identifier</i>	error identifier Type: char
in	<i>format</i>	string with error message printf-style format
	...	arguments to be formatted

Returns

void

Definition at line 113 of file amici.cpp.

9.1.4.2 printWarnMsgIdAndTxt()

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

printErrMsgIdAndTxt prints a specified warning message associated to the specified identifier

Parameters

in	<i>identifier</i>	warning identifier Type: char
in	<i>format</i>	string with error message printf-style format
	...	arguments to be formatted

Returns

void

Definition at line 134 of file amici.cpp.

9.1.4.3 runAmiciSimulation() [1/2]

```
std::unique_ptr< ReturnData > runAmiciSimulation (
    Solver & solver,
    const ExpData * edata,
    Model & model )
```

runAmiciSimulation is the core integration routine. It initializes the solver and runs the forward and backward problem.

Parameters

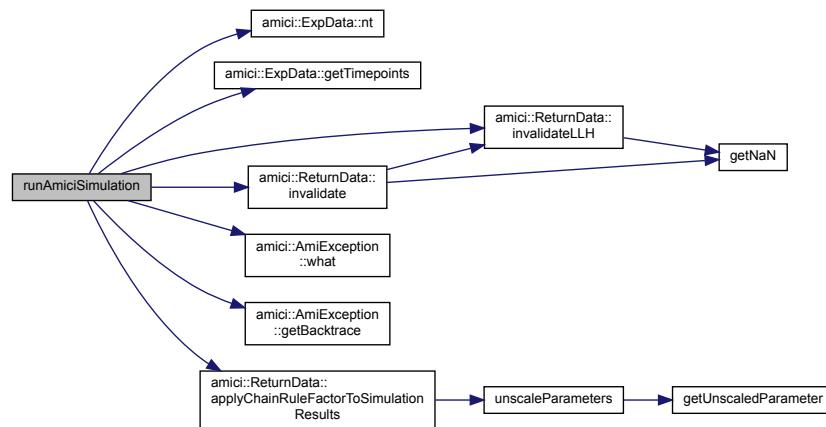
<i>solver</i>	Solver instance
<i>edata</i>	pointer to experimental data object
<i>model</i>	model specification object

Returns

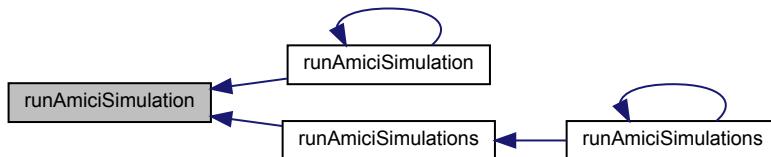
rdata pointer to return data object

Definition at line 42 of file amici.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.4 runAmiciSimulations() [1/2]**

```

std::vector< std::unique_ptr< ReturnData > > runAmiciSimulations (
    Solver const & solver,
    const std::vector< ExpData * > & edatas,
    Model const & model,
    int num_threads )
  
```

runAmiciSimulations does the same as runAmiciSimulation, but for multiple `ExpData` instances.

Parameters

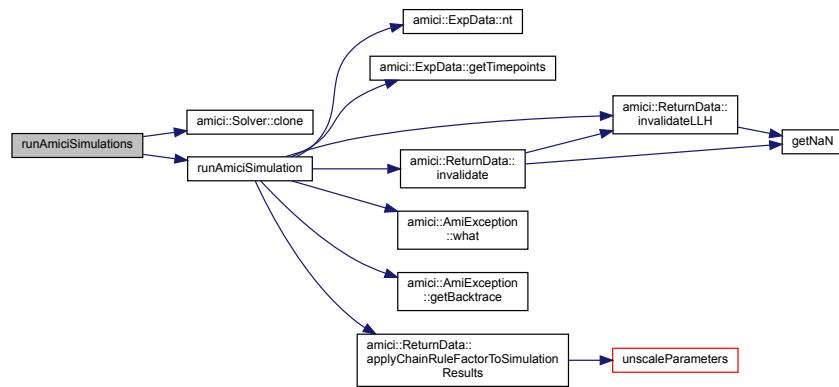
<code>solver</code>	<code>Solver</code> instance
<code>edatas</code>	experimental data objects
<code>model</code>	model specification object
<code>num_threads</code>	number of threads for parallel execution

Returns

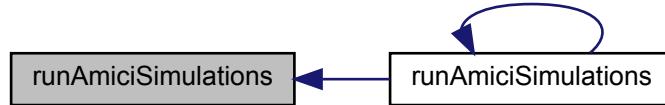
vector of pointers to return data objects

Definition at line 146 of file amici.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.5 amici_dgemv()**

```

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

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

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

Returns

void

Definition at line 73 of file cblas.cpp.

9.1.4.6 amici_dgemm()

```
void amici_dgemm (
    BLASLayout layout,
    BLASTranspose TransA,
    BLASTranspose 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 $C = \alpha A \cdot B + \beta C$ with $A: [MxK]$ $B:[KxN]$ $C:[MxN]$

Parameters

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

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

Parameters

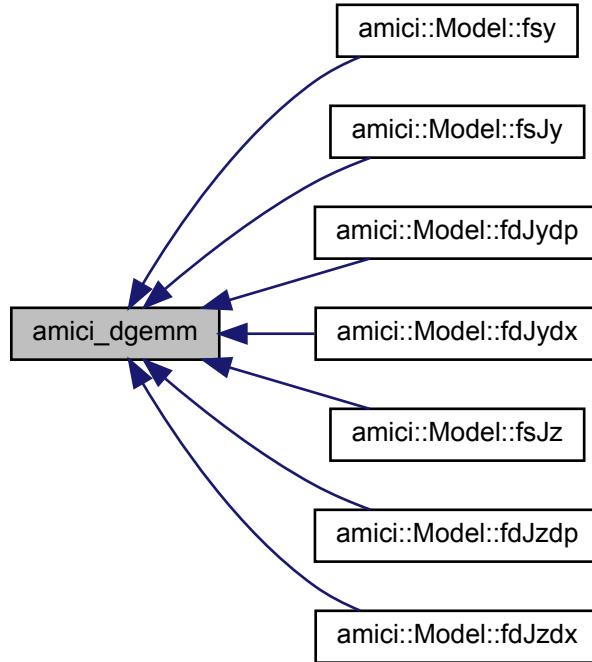
<i>layout</i>	memory layout.
<i>TransA</i>	flag indicating whether A should be transposed before multiplication
<i>TransB</i>	flag indicating whether B should be transposed before multiplication
<i>M</i>	number of rows in A/C
<i>N</i>	number of columns in B/C
<i>K</i>	number of rows in B, number of columns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

Returns

void

Definition at line 44 of file cblas.cpp.

Here is the caller graph for this function:



9.1.4.7 amici_daxpy()

```
void amici_daxpy (
    int n,
    double alpha,
    const double * x,
    const int incx,
    double * y,
    int incy )
```

Parameters

<i>n</i>	number of elements in y
<i>alpha</i>	scalar coefficient of x
<i>x</i>	vector of length n*incx
<i>incx</i>	x stride
<i>y</i>	vector of length n*incy
<i>incy</i>	y stride

Definition at line 90 of file cblas.cpp.

9.1.4.8 setModelData()

```
void setModelData (
    const mxArray * prhs[],
    int nrhs,
    Model & model )
```

Parameters

in	<i>prhs</i>	pointer to the array of input arguments Type: mxArray
in	<i>nrhs</i>	number of elements in prhs
in, out	<i>model</i>	model to update

Definition at line 389 of file interface_matlab.cpp.

Here is the call graph for this function:



9.1.4.9 setSolverOptions()

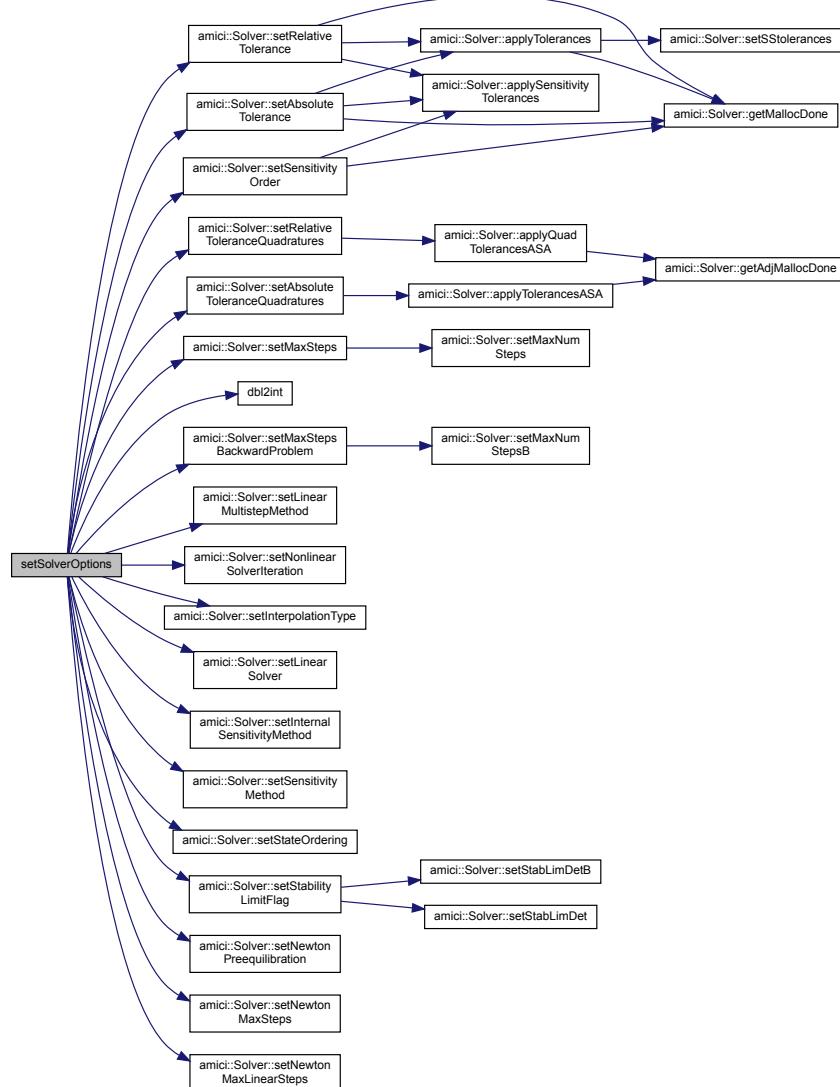
```
void setSolverOptions (
    const mxArray * prhs[],
    int nrhs,
    Solver & solver )
```

Parameters

in	<i>prhs</i>	pointer to the array of input arguments Type: mxArray
in	<i>nrhs</i>	number of elements in prhs
in, out	<i>solver</i>	solver to update

Definition at line 304 of file interface_matlab.cpp.

Here is the call graph for this function:



9.1.4.10 setupReturnData()

```
ReturnDataMatlab* amici::setupReturnData (
    mxArray * plhs[],
    int nlhs )
```

setupReturnData initialises the return data struct

Parameters

in	<i>plhs</i>	user input Type: mxArray
in	<i>nlhs</i>	number of elements in plhs Type: mxArray

Returns

rdata: return data struct

Type: *ReturnData**9.1.4.11 expDataFromMatlabCall()**

```
std::unique_ptr< ExpData > expDataFromMatlabCall (
    const mxArray * prhs[],
    const Model & model )
```

`expDataFromMatlabCall` initialises the experimental data struct

Parameters

in	<i>prhs</i>	user input Type: *mxArray
----	-------------	-------------------------------------

Returns

edata: experimental data struct

Type: *ExpData

`expDataFromMatlabCall` parses the experimental data from the matlab call and writes it to an [ExpData](#) class object

Parameters

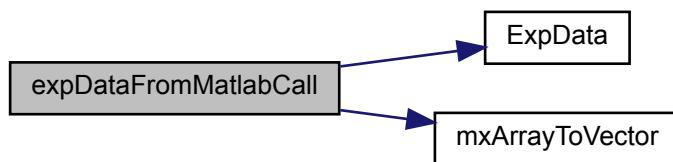
<i>prhs</i>	pointer to the array of input arguments
<i>model</i>	pointer to the model object, this is necessary to perform dimension checks

Returns

edata pointer to experimental data object

Definition at line 179 of file `interface_matlab.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.12 checkFinite()

```
int checkFinite (
    const int N,
    const realtype * array,
    const char * fun )
```

Checks the values in an array for NaNs and Infs

Parameters

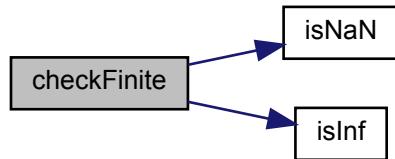
<i>N</i>	number of elements in array
<i>array</i>	array
<i>fun</i>	name of calling function

Returns

AMICI_RECOVERABLE_ERROR if a NaN/Inf value was found, AMICI_SUCCESS otherwise

Definition at line 17 of file misc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.13 unscaleParameters() [1/2]

```
void unscaleParameters (
    const double * bufferScaled,
    const ParameterScaling * pscale,
    int n,
    double * bufferUnscaled )
```

Parameters

<i>bufferScaled</i>	scaled parameters
<i>pscale</i>	parameter scaling
<i>n</i>	number of elements in bufferScaled, pscale and bufferUnscaled
<i>bufferUnscaled</i>	unscaled parameters are written to the array

Returns

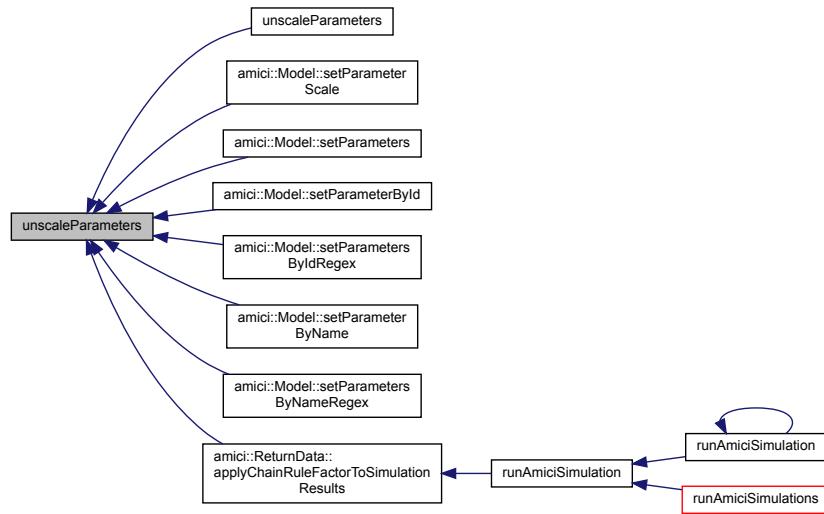
status flag indicating success of execution
Type: int

Definition at line 46 of file misc.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.14 unscaleParameters() [2/2]

```
void unscaleParameters (
    std::vector< double > const & bufferScaled,
    std::vector< ParameterScaling > const & pscale,
    std::vector< double > & bufferUnscaled )
```

All vectors must be of same length

Parameters

<i>bufferScaled</i>	scaled parameters
<i>pscale</i>	parameter scaling
<i>bufferUnscaled</i>	unscaled parameters are written to the array

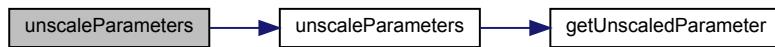
Returns

status flag indicating success of execution

Type: int

Definition at line 54 of file misc.cpp.

Here is the call graph for this function:



9.1.4.15 getUnscaledParameter()

```
double getUnscaledParameter (
    double scaledParameter,
    ParameterScaling scaling )
```

Parameters

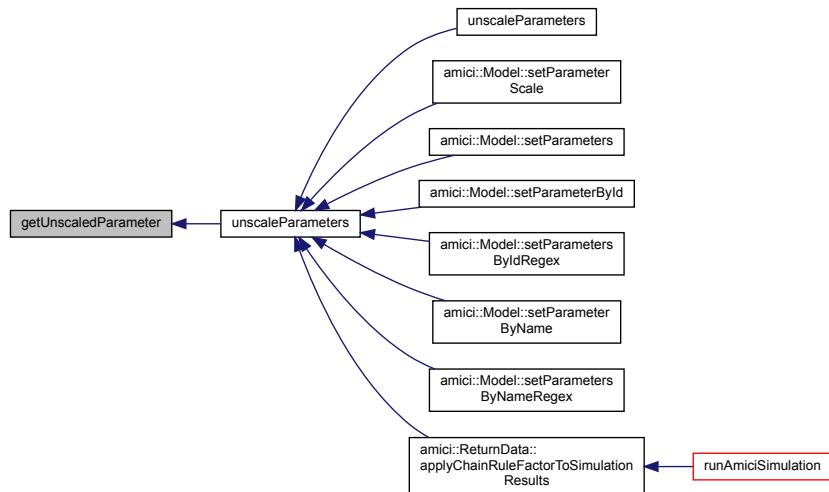
<i>scaledParameter</i>	scaled parameter
<i>scaling</i>	parameter scaling

Returns

Unscaled parameter

Definition at line 32 of file misc.cpp.

Here is the caller graph for this function:



9.1.4.16 operator==([1/2]

```
bool operator== (
    const Model & a,
    const Model & b )
```

Parameters

<i>a</i>	first model instance
<i>b</i>	second model instance

Returns

`equality`

Definition at line 1297 of file `model.cpp`.

9.1.4.17 `getReturnDataMatlabFromAmiciCall()`

```
mxArray * getReturnDataMatlabFromAmiciCall (
    ReturnData const * rdata )
```

generates matlab mxArray from a [ReturnData](#) object

Parameters

<code>rdata</code>	ReturnDataObject
--------------------	------------------

Returns

`rdatamatlab` ReturnDataObject stored as matlab compatible data

generates matlab mxArray from a [ReturnData](#) object

Parameters

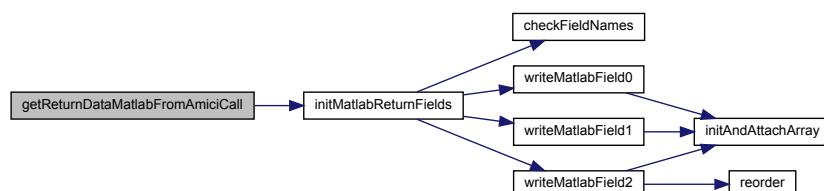
<code>rdata</code>	ReturnDataObject
--------------------	------------------

Returns

`rdatamatlab` ReturnDataObject stored as matlab compatible data

Definition at line 7 of file `returndata_matlab.cpp`.

Here is the call graph for this function:

**9.1.4.18 `initMatlabReturnFields()`**

```
mxArray * initMatlabReturnFields (
    ReturnData const * rdata )
```

allocates and initialises solution mxArray with the corresponding fields

Parameters

<code>rdata</code>	ReturnDataObject
--------------------	------------------

Returns

Solution mxArray

allocates and initialises solution mxArray with the corresponding fields

Parameters

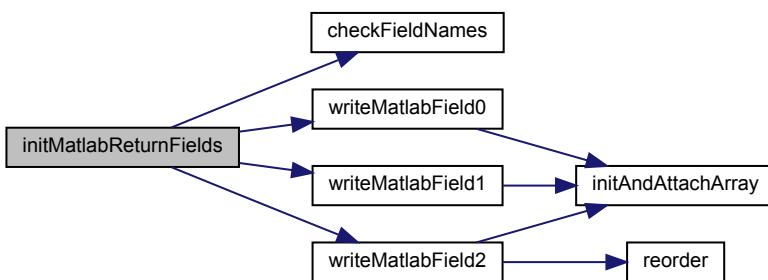
<code>rdata</code>	ReturnDataObject
--------------------	------------------

Returns

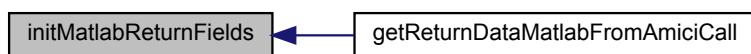
Solution mxArray

Definition at line 18 of file returndata_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.19 initMatlabDiagnosisFields()**

```

mxArray * initMatlabDiagnosisFields (
    ReturnData const * rdata )
  
```

allocates and initialises diagnosis mxArray with the corresponding fields

Parameters

<i>rdata</i>	ReturnDataObject
--------------	------------------

Returns

Diagnosis mxArray

allocates and initialises diagnosis mxArray with the corresponding fields

Parameters

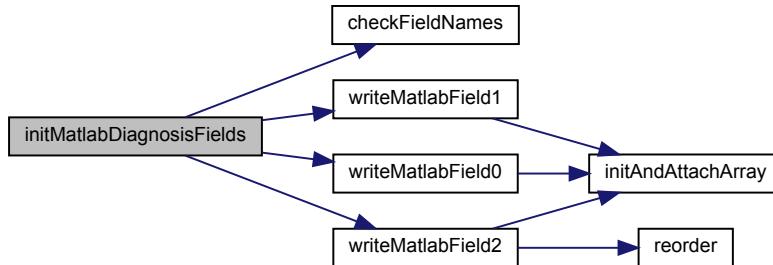
<i>rdata</i>	ReturnDataObject
--------------	------------------

Returns

Diagnosis mxArray

Definition at line 116 of file returndata_matlab.cpp.

Here is the call graph for this function:

**9.1.4.20 writeMatlabField0()**

```

void writeMatlabField0 (
    mxArray * matlabStruct,
    const char * fieldName,
    T fieldData )
  
```

initialise vector and attach to the field

Parameters

<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field

initialise vector and attach to the field

Parameters

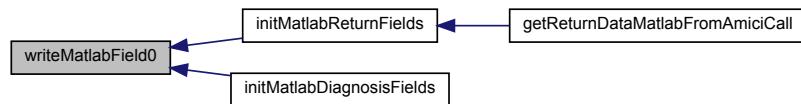
<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field

Definition at line 181 of file returndata_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.21 writeMatlabField1()

```

void writeMatlabField1 (
    mxArray * matlabStruct,
    const char * fieldName,
    std::vector< T > fieldData,
    const int dim0 )
  
```

initialise vector and attach to the field

Parameters

<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of elements in the vector

initialise vector and attach to the field

Parameters

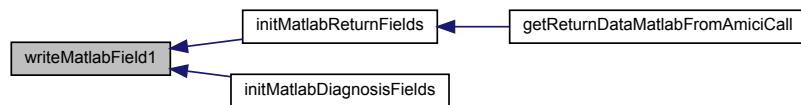
<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of elements in the vector

Definition at line 199 of file returndata_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.22 writeMatlabField2()

```

void writeMatlabField2 (
    mxArray * matlabStruct,
    const char * fieldName,
    std::vector< T > fieldData,
    int dim0,
    int dim1,
    std::vector< int > perm )
  
```

initialise matrix, attach to the field and write data

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of rows in the tensor
<i>dim1</i>	Number of columns in the tensor
<i>perm</i>	reordering of dimensions (i.e., transposition)

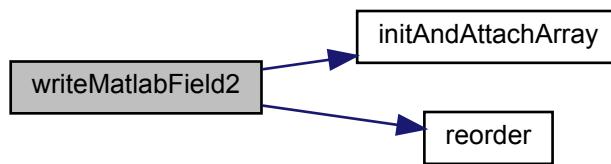
initialise matrix, attach to the field and write data

Parameters

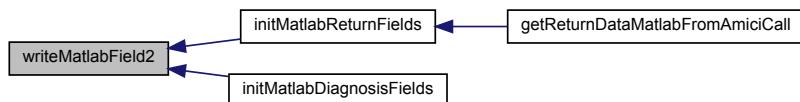
<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of rows in the tensor
<i>dim1</i>	Number of columns in the tensor
<i>perm</i>	reordering of dimensions (i.e., transposition)

Definition at line 221 of file returndata_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.23 writeMatlabField3()

```

void writeMatlabField3 (
    mxArray * matlabStruct,
    const char * fieldName,
    std::vector< T > fieldData,
    int dim0,
    int dim1,
    int dim2,
    std::vector< int > perm )
  
```

initialise 3D tensor, attach to the field and write data

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	number of rows in the tensor
<i>dim1</i>	number of columns in the tensor
<i>dim2</i>	number of elements in the third dimension of the tensor
<i>perm</i>	reordering of dimensions

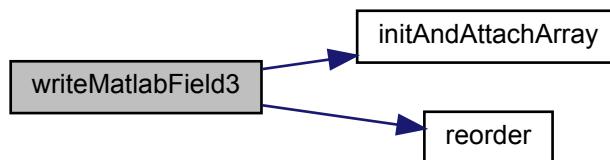
initialise 3D tensor, attach to the field and write data

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	number of rows in the tensor
<i>dim1</i>	number of columns in the tensor
<i>dim2</i>	number of elements in the third dimension of the tensor
<i>perm</i>	reordering of dimensions

Definition at line 254 of file returndata_matlab.cpp.

Here is the call graph for this function:

**9.1.4.24 writeMatlabField4()**

```

void writeMatlabField4 (
    mxArray * matlabStruct,
    const char * fieldName,
    std::vector< T > fieldData,
    int dim0,
    int dim1,
    int dim2,
    int dim3,
    std::vector< int > perm )
  
```

initialise 4D tensor, attach to the field and write data

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	number of rows in the tensor
<i>dim1</i>	number of columns in the tensor
<i>dim2</i>	number of elements in the third dimension of the tensor
<i>dim3</i>	number of elements in the fourth dimension of the tensor
<i>perm</i>	reordering of dimensions

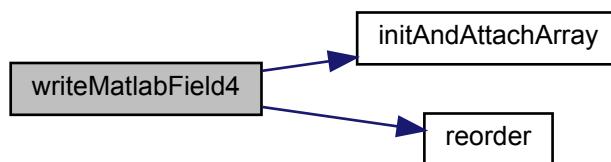
initialise 4D tensor, attach to the field and write data

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	number of rows in the tensor
<i>dim1</i>	number of columns in the tensor
<i>dim2</i>	number of elements in the third dimension of the tensor
<i>dim3</i>	number of elements in the fourth dimension of the tensor
<i>perm</i>	reordering of dimensions

Definition at line 290 of file `returndata_matlab.cpp`.

Here is the call graph for this function:



9.1.4.25 `initAndAttachArray()`

```
double * initAndAttachArray (
    mxArray * matlabStruct,
    const char * fieldName,
    std::vector< mwSize > dim )
```

initialises the field *fieldName* in *matlabStruct* with dimension *dim*

Parameters

<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>dim</i>	vector of field dimensions

Returns

Pointer to field data

Initialises the field *fieldName* in *matlabStruct* with dimension *dim*

Parameters

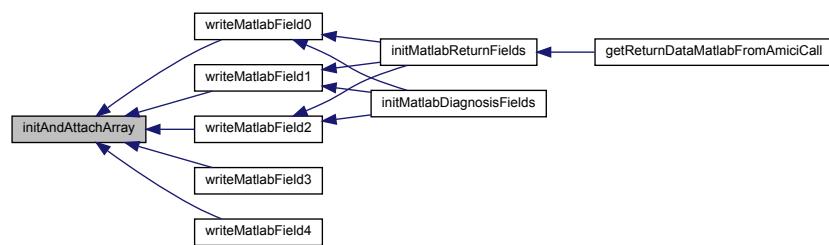
<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>dim</i>	vector of field dimensions

Returns

Pointer to field data

Definition at line 328 of file *returndata_matlab.cpp*.

Here is the caller graph for this function:

**9.1.4.26 checkFieldNames()**

```
void checkFieldNames (
    const char ** fieldNames,
    const int fieldCount )
```

checks whether *fieldNames* was properly allocated

Parameters

<i>fieldNames</i>	array of field names
<i>fieldCount</i>	expected number of fields in <i>fieldNames</i>

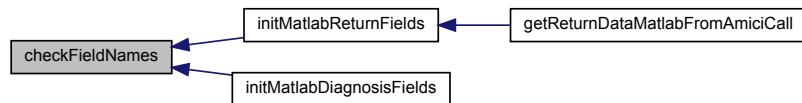
checks whether fieldNames was properly allocated

Parameters

<i>fieldNames</i>	array of field names
<i>fieldCount</i>	expected number of fields in fieldNames

Definition at line 349 of file returndata_matlab.cpp.

Here is the caller graph for this function:



9.1.4.27 reorder()

```
std::vector< T > reorder (
    const std::vector< T > input,
    const std::vector< int > order )
```

template function that reorders elements in a std::vector

Parameters

<i>input</i>	unordered vector
<i>order</i>	dimension permutation

Returns

Reordered vector

template function that reorders elements in a std::vector

Parameters

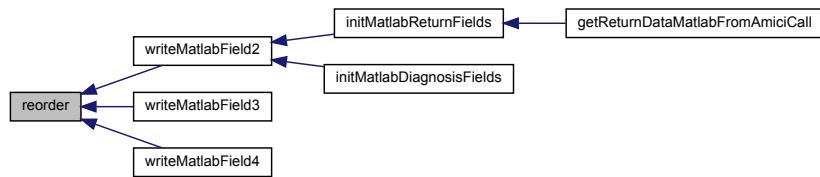
<i>input</i>	unordered vector
<i>order</i>	dimension permutation

Returns

Reordered vector

Definition at line 362 of file returndata_matlab.cpp.

Here is the caller graph for this function:



9.1.4.28 serializeToChar()

```
char* amici::serializeToChar (
    T const & data,
    int * size )
```

Serialize object to char array

Parameters

<i>data</i>	input object
<i>size</i>	maximum char length

Returns

The object serialized as char

Definition at line 172 of file serialization.h.

9.1.4.29 deserializeFromChar()

```
T amici::deserializeFromChar (
    const char * buffer,
    int size )
```

Deserialize object that has been serialized using serializeToChar

Parameters

<i>buffer</i>	serialized object
<i>size</i>	length of buffer

Returns

The deserialized object

Definition at line 205 of file serialization.h.

9.1.4.30 serializeToString()

```
std::string amici::serializeToString (
    T const & data )
```

Serialize object to string

Parameters

<i>data</i>	input object
-------------	--------------

Returns

The object serialized as string

Definition at line 230 of file serialization.h.

9.1.4.31 serializeToStdVec()

```
std::vector<char> amici::serializeToStdVec (
    T const & data )
```

Serialize object to std::vector<char>

Parameters

<i>data</i>	input object
-------------	--------------

Returns

The object serialized as std::vector<char>

Definition at line 256 of file serialization.h.

9.1.4.32 deserializeFromString()

```
T amici::deserializeFromString (
    std::string const & serialized )
```

Deserialize object that has been serialized using serializeToString

Parameters

<i>serialized</i>	serialized object
-------------------	-------------------

Returns

The deserialized object

Definition at line 283 of file serialization.h.

9.1.4.33 operator==() [2/2]

```
bool operator== (
    const Solver & a,
    const Solver & b )
```

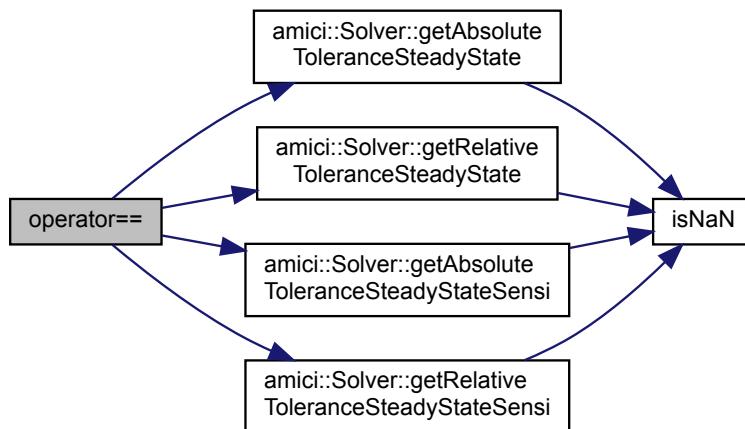
Parameters

<i>a</i>	
<i>b</i>	

Returns

Definition at line 378 of file solver.cpp.

Here is the call graph for this function:



9.1.4.34 `spline()` [1/2]

```
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 $b[i]$, $c[i]$, $d[i]$, $i = 0, 1, \dots, n-1$ 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 $x[i] \leq xx \leq x[i+1]$

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

Parameters

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

Return values

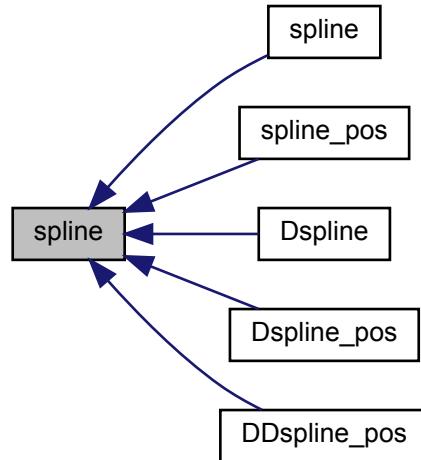
0	normal return
1	less than two data points; cannot interpolate
2	$x[]$ are not in ascending order

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 $[0 \dots n-1]$.
- 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 required 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 16 of file spline.cpp.

Here is the caller graph for this function:



9.1.4.35 seval()

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

$S(xx) = y[i] + b[i] * w + c[i] * w^{**2} + d[i] * w^{**3}$ where $w = u - x[i]$ and $x[i] \leq u \leq x[i+1]$ 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	<i>n</i>	The number of data points or knots ($n \geq 2$)
in	<i>u</i>	the abscissa at which the spline is to be evaluated
in	<i>x[]</i>	the abscissas of the knots in strictly increasing order
in	<i>y[]</i>	the ordinates of the knots
in	<i>b</i>	array of spline coefficients computed by spline() .
in	<i>c</i>	array of spline coefficients computed by spline() .
in	<i>d</i>	array of spline coefficients computed by spline() .

Returns

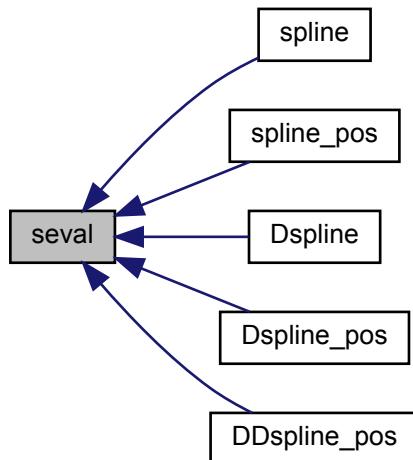
the value of the spline function at u

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 195 of file spline.cpp.

Here is the caller graph for this function:

**9.1.4.36 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 $x[i] \leq u \leq x[i+1]$

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	<i>n</i>	the number of data points or knots (<i>n</i> >= 2)
in	<i>u</i>	the abscissa at which the spline is to be evaluated
in	<i>x[]</i>	the abscissas of the knots in strictly increasing order
in	<i>y[]</i>	the ordinates of the knots
in	<i>b</i>	array of spline coefficients computed by spline() .
in	<i>c</i>	array of spline coefficients computed by spline() .
in	<i>d</i>	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 256 of file `spline.cpp`.

9.1.4.37 log()

```
double log (
    double x )
```

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

Parameters

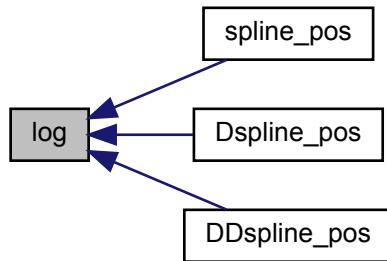
<i>x</i>	argument
----------	----------

Returns

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

Definition at line 62 of file `symbolic_functions.cpp`.

Here is the caller graph for this function:



9.1.4.38 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 77 of file symbolic_functions.cpp.

9.1.4.39 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 92 of file symbolic_functions.cpp.

9.1.4.40 min()

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

c implementation of matlab function min

Parameters

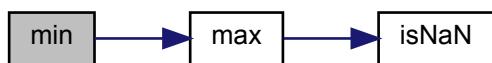
<i>a</i>	value1
<i>b</i>	value2
<i>c</i>	bogus parameter to ensure correct parsing as a function

Returns

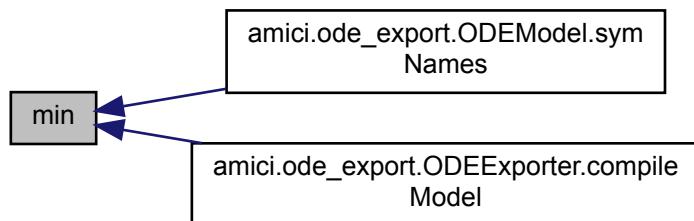
```
if(a < b) then a else b
```

Definition at line 149 of file symbolic_functions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



9.1.4.41 Dmin()

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

c implementation of matlab function max

Parameters

<i>id</i>	argument index for differentiation
<i>a</i>	value1
<i>b</i>	value2
<i>c</i>	bogus parameter to ensure correct parsing as a function

Returns

```
id == 1: if(a > b) then 1 else 0  
id == 2: if(a > b) then 0 else 1
```

Definition at line 191 of file symbolic_functions.cpp.

Here is the call graph for this function:



9.1.4.42 max()

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

c implementation of matlab function max

Parameters

<i>a</i>	value1
<i>b</i>	value2
<i>c</i>	bogus parameter to ensure correct parsing as a function

Returns

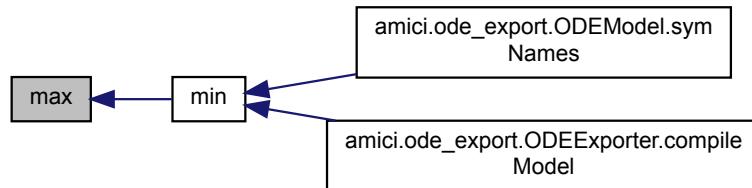
```
if(a > b) then a else b
```

Definition at line 128 of file symbolic_functions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.43 Dmax()**

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

parameter derivative of c implementation of matlab function max

Parameters

<i>id</i>	argument index for differentiation
<i>a</i>	value1
<i>b</i>	value2
<i>c</i>	bogus parameter to ensure correct parsing as a function

Returns

```
id == 1: if(a > b) then 1 else 0
id == 2: if(a > b) then 0 else 1
```

Definition at line 164 of file symbolic_functions.cpp.

Here is the caller graph for this function:

**9.1.4.44 pos_pow()**

```
double pos_pow (
    double base,
    double exponent )
```

specialized pow functions that assumes positivity of the first argument

Parameters

<i>base</i>	base
<i>exponent</i>	exponent

Returns

```
pow(max(base,0.0),exponent)
```

Definition at line 203 of file symbolic_functions.cpp.

9.1.4.45 isNaN()

```
int isNaN (
    double what )
```

c++ interface to the isNaN function

Parameters

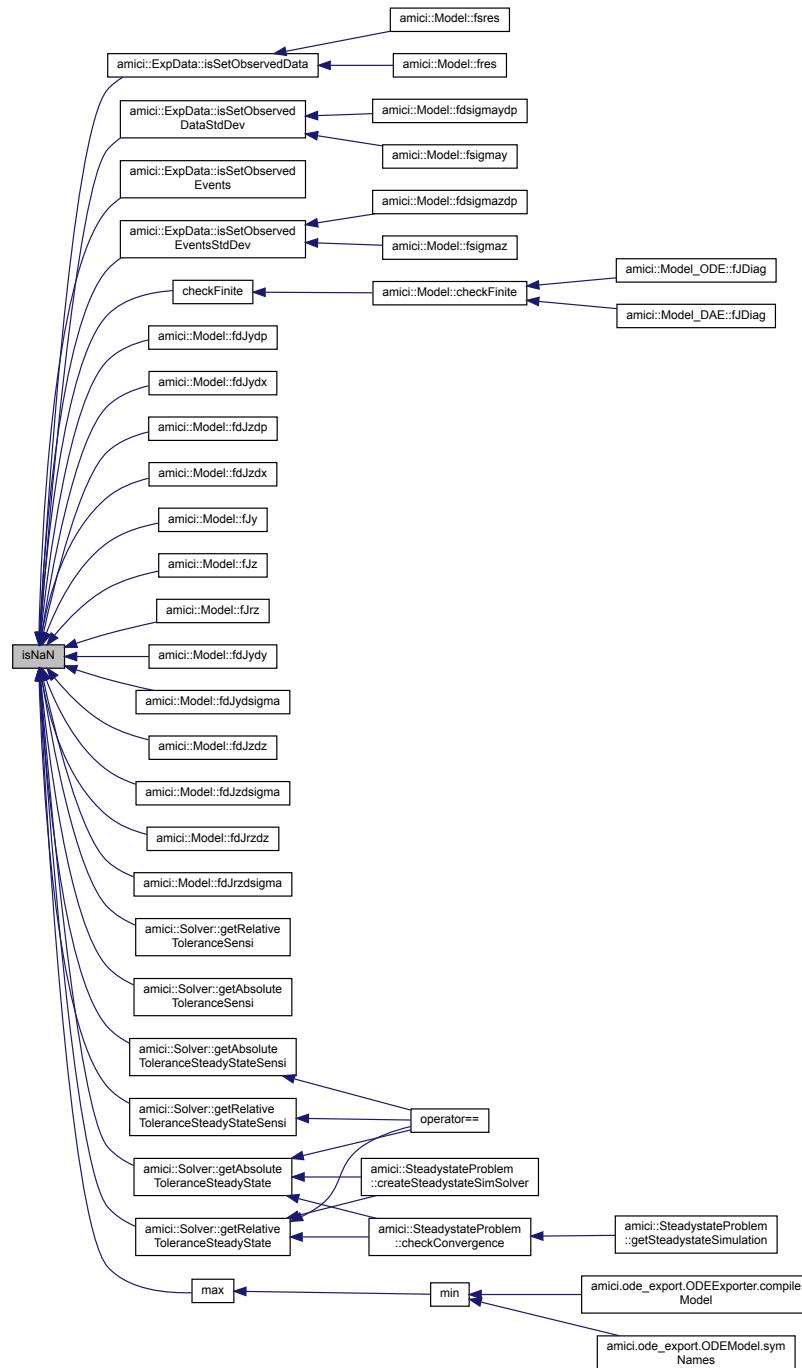
<i>what</i>	argument
-------------	----------

Returns

```
isnan(what)
```

Definition at line 35 of file symbolic_functions.cpp.

Here is the caller graph for this function:



9.1.4.46 isInf()

```
int isInf (
    double what )
```

c++ interface to the isinf function

Parameters

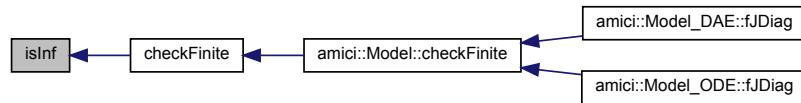
what	argument
------	----------

Returns

isnan(what)

Definition at line 44 of file symbolic_functions.cpp.

Here is the caller graph for this function:



9.1.4.47 getNaN()

```
double getNaN ( )
```

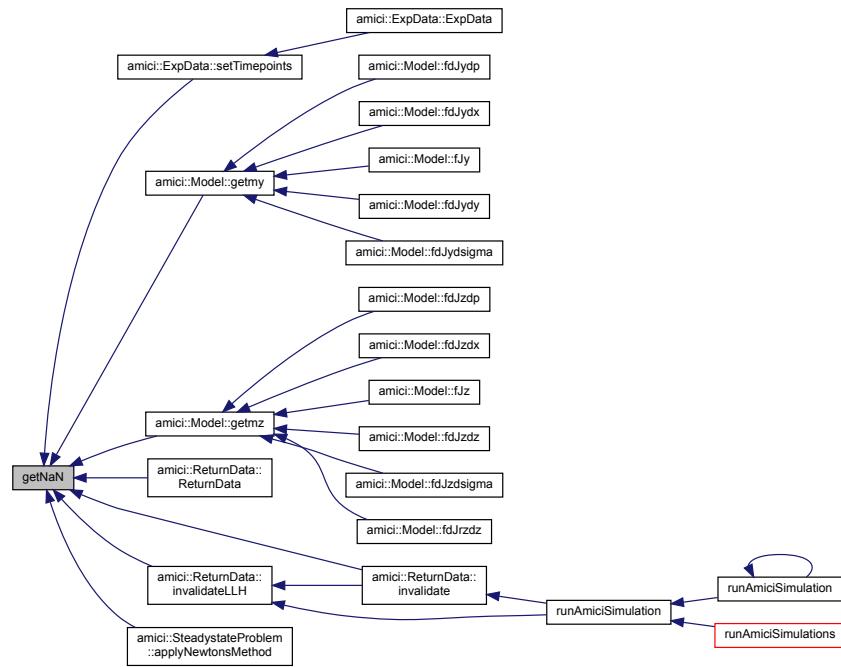
function returning nan

Returns

NaN

Definition at line 52 of file symbolic_functions.cpp.

Here is the caller graph for this function:



9.1.4.48 sign()

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

c implementation of matlab function sign

Parameters

x	argument
---	----------

Returns

0

Definition at line 107 of file symbolic_functions.cpp.

9.1.4.49 spline() [2/2]

```
double 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

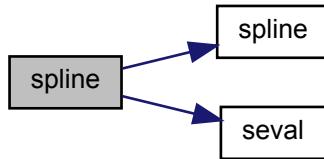
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

Returns

`spline(t)`

Definition at line 222 of file `symbolic_functions.cpp`.

Here is the call graph for this function:

**9.1.4.50 `spline_pos()`**

```
double 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

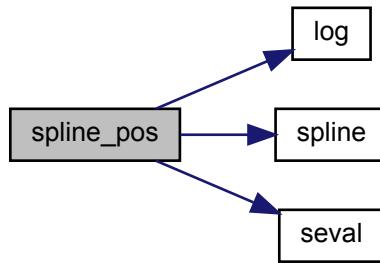
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

Returns

`spline(t)`

Definition at line 273 of file `symbolic_functions.cpp`.

Here is the call graph for this function:



9.1.4.51 Dspline()

```
double 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

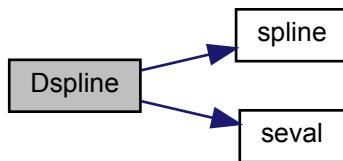
<i>id</i>	index of node to which the derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

Returns

```
dsplinedp(t)
```

Definition at line 326 of file symbolic_functions.cpp.

Here is the call graph for this function:

**9.1.4.52 Dspline_pos()**

```
double 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

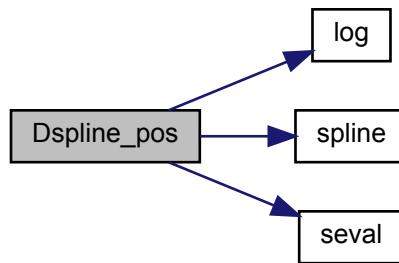
<code>id</code>	index of node to which the derivative of the corresponding spline coefficient should be computed
<code>t</code>	point at which the spline should be evaluated
<code>num</code>	number of spline nodes

Returns

```
dsplinedp(t)
```

Definition at line 382 of file symbolic_functions.cpp.

Here is the call graph for this function:

**9.1.4.53 DDspline()**

```
double 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

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

Returns

```
ddspline(t)
```

Definition at line 448 of file symbolic_functions.cpp.

9.1.4.54 DDspline_pos()

```
double 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

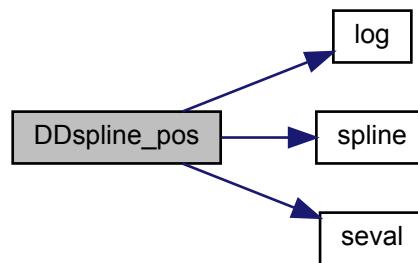
<i>id1</i>	index of node to which the first derivative of the corresponding spline coefficient should be computed
<i>id2</i>	index of node to which the second derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number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

Returns

ddsspline(t)

Definition at line 468 of file symbolic_functions.cpp.

Here is the call graph for this function:



9.1.4.55 runAmiciSimulation() [2/2]

```
def amici.runAmiciSimulation (
    model,
    solver,
    edata = None )
```

Parameters

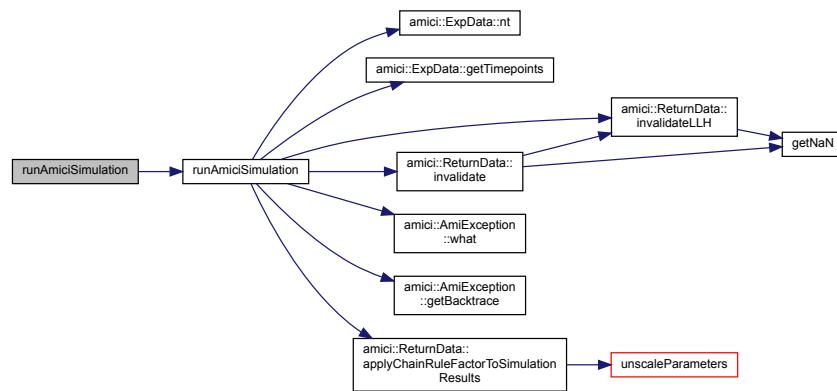
<i>model</i>	Model instance
<i>solver</i>	Solver instance, must be generated from Model.getSolver()
<i>edata</i>	ExpData instance (optional)

Returns

[ReturnData](#) object with simulation results

Definition at line 93 of file `__init__.py`.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.56 ExpData()**

```
def amici.ExpData (
    args )
```

Parameters

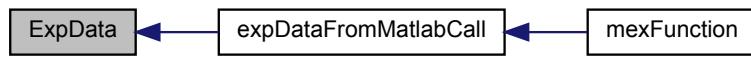
<i>args</i>	arguments
-------------	-----------

Returns

[ExpData](#) Instance

Definition at line 112 of file `__init__.py`.

Here is the caller graph for this function:

**9.1.4.57 runAmiciSimulations()** [2/2]

```
def amici.runAmiciSimulations (
    model,
    solver,
    edata_list,
    num_threads = 1 )
```

Parameters

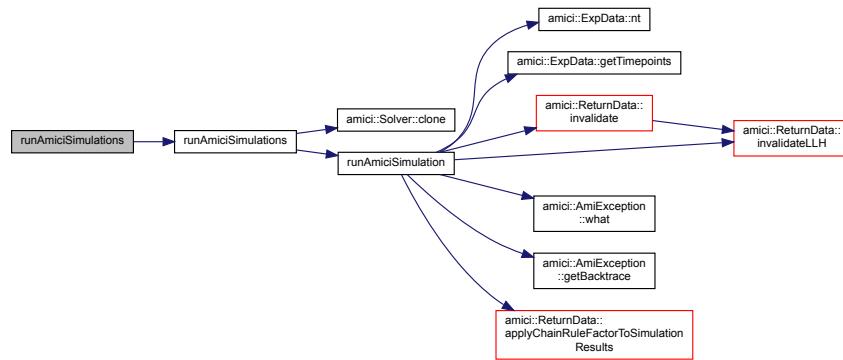
<i>model</i>	Model instance
<i>solver</i>	Solver instance, must be generated from Model.getSolver()
<i>edata_list</i>	list of ExpData instances
<i>num_threads</i>	number of threads to use (only used if compiled with openmp)

Returns

list of [ReturnData](#) objects with simulation results

Definition at line 140 of file `__init__.py`.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.58 dbl2int()**

```
int dbl2int (
    const double x )
```

conversion from double to int with checking for loss of data

Parameters

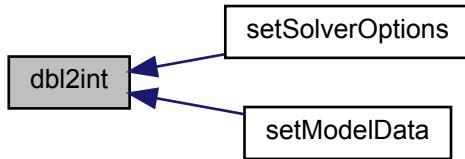
x	input
---	-------

Returns

int_x casted value

Definition at line 298 of file interface_matlab.cpp.

Here is the caller graph for this function:



9.1.4.59 amici_blasCBlasTransToBlasTrans()

```
char amici::amici_blasCBlasTransToBlasTrans (
    BLASTranspose trans )
```

amici_blasCBlasTransToBlasTrans translates AMICI_BLAS_TRANSPOSE values to CBlas readable strings

Parameters

<i>trans</i>	flag indicating transposition and complex conjugation
--------------	---

Returns

cblatrans CBlas readable CHAR indicating transposition and complex conjugation

Definition at line 52 of file interface_matlab.cpp.

9.1.4.60 mxArrayToVector()

```
std::vector<realtyp> amici::mxArrayToVector (
    const mxArray * array,
    int length )
```

conversion from mxArray to vector<realtyp>

Parameters

<i>array</i>	Matlab array to create vector from
<i>length</i>	Number of elements in array

Returns

`std::vector< std::string >` with data from array

Definition at line 166 of file `interface_matlab.cpp`.

Here is the caller graph for this function:

**9.1.4.61 getValueById()**

```

realtype amici::getValueById (
    std::vector< std::string > const & ids,
    std::vector< realtype > const & values,
    std::string const & id,
    const char * variable_name,
    const char * id_name )
  
```

Parameters

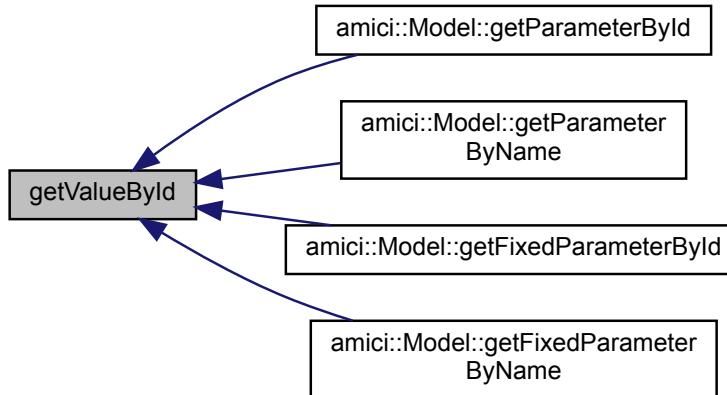
<i>ids</i>	vector of name/ids of (fixed)Parameters
<i>values</i>	values of the (fixed)Parameters
<i>id</i>	name/id to look for in the vector
<i>variable_name</i>	string indicating what variable we are lookin at
<i>id_name</i>	string indicating whether name or id was specified

Returns

value of the selected parameter

Definition at line 323 of file `model.cpp`.

Here is the caller graph for this function:



9.1.4.62 `setValueById()`

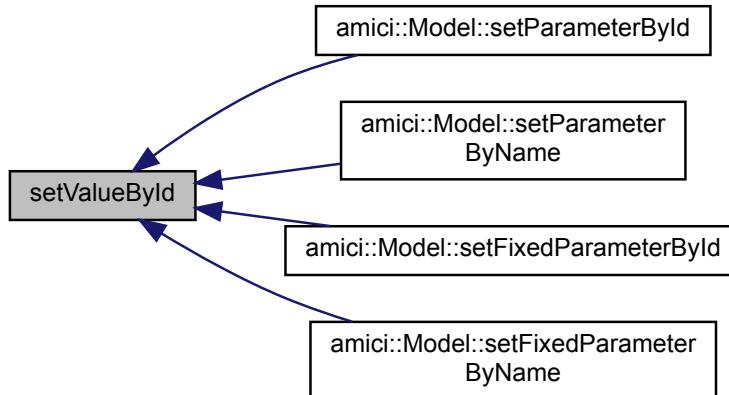
```
void amici::setValueById (
    std::vector< std::string > const & ids,
    std::vector< realltype > & values,
    realltype value,
    std::string const & id,
    const char * variable_name,
    const char * id_name )
```

Parameters

<code>ids</code>	vector of names/ids of (fixed)Parameters
<code>values</code>	values of the (fixed)Parameters
<code>value</code>	for the selected parameter
<code>id</code>	name/id to look for in the vector
<code>variable_name</code>	string indicating what variable we are lookin at
<code>id_name</code>	string indicating whether name or id was specified

Definition at line 341 of file `model.cpp`.

Here is the caller graph for this function:



9.1.4.63 setValueByldRegex()

```
int amici::setValueByldRegex (
    std::vector< std::string > const & ids,
    std::vector< realltype > & values,
    realltype value,
    std::string const & regex,
    const char * variable_name,
    const char * id_name )
```

Parameters

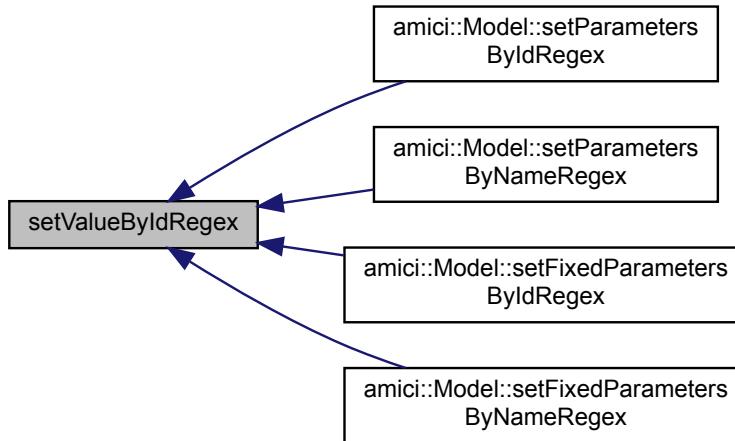
<i>ids</i>	vector of names/ids of (fixed)Parameters
<i>values</i>	values of the (fixed)Parameters
<i>value</i>	for the selected parameter
<i>regex</i>	string according to which names/ids are to be matched
<i>variable_name</i>	string indicating what variable we are lookin at
<i>id_name</i>	string indicating whether name or id was specified

Returns

number of matched names/ids

Definition at line 360 of file model.cpp.

Here is the caller graph for this function:



9.1.5 Variable Documentation

9.1.5.1 errMsgIdAndTxt

```
msgIdAndTxtFp errMsgIdAndTxt = &printErrMsgIdAndTxt
```

`errMsgIdAndTxt` is a function pointer for `printErrMsgIdAndTxt`

Definition at line 37 of file `amici.cpp`.

9.1.5.2 warnMsgIdAndTxt

```
msgIdAndTxtFp warnMsgIdAndTxt = &printWarnErrMsgIdAndTxt
```

`warnMsgIdAndTxt` is a function pointer for `printWarnErrMsgIdAndTxt`

Definition at line 13 of file `model_dae.h`.

9.1.5.3 pi

```
constexpr double pi = 3.14159265358979323846
```

MS definition of PI and other constants

Definition at line 13 of file `defines.h`.

9.1.5.4 amici_path

amici_path

Initial value:

```
1 = os.path.abspath(os.path.join(
2     os.path.dirname(__file__), '...', ...))
```

Definition at line 55 of file `__init__.py`.

9.2 amici.ode_export Namespace Reference

The C++ ODE export module for python.

Classes

- class [Constant](#)
A [Constant](#) is a fixed variable in the model with respect to which sensitivities cannot be computed, abbreviated by k
- class [Expression](#)
An [Expression](#) is a recurring elements in symbolic formulas.
- class [LogLikelihood](#)
A [LogLikelihood](#) defines the distance between measurements and experiments for a particular observable.
- class [ModelQuantity](#)
Base class for model components.
- class [Observable](#)
An [Observable](#) links model simulations to experimental measurements, abbreviated by y
- class [ODEExporter](#)
The [ODEExporter](#) class generates AMICI C++ files for ODE model as defined in symbolic expressions.
- class [ODEModel](#)
An [ODEModel](#) defines an Ordinary Differential Equation as set of [ModelQuantities](#).
- class [Parameter](#)
A [Parameter](#) is a free variable in the model with respect to which sensitivities may be computed, abbreviated by p
- class [SigmaY](#)
A Standard Deviation [SigmaY](#) rescales the distance between simulations and measurements when computing residuals, abbreviated by sigmay
- class [State](#)
A [State](#) variable defines an entity that evolves with time according to the provided time derivative, abbreviated by x
- class [TemplateAmici](#)
Template format used in AMICI (see `string.template` for more details).

Functions

- def [var_in_function_signature](#) (name, varname)
Checks if the values for a symbolic variable is passed in the signature of a function.
- def [getSymbolicDiagonal](#) (matrix)
Get symbolic matrix with diagonal of matrix `matrix`.
- def [applyTemplate](#) (sourceFile, targetFile, templateData)
Load source file, apply template substitution as provided in `templateData` and save as `targetFile`.
- def [sanitize_basic_sympy](#) (basic)
Strips pysb info from the `sympy.Basic` object.

Variables

- `pysb = None`
`pysb import dummy`
- dictionary `functions`
prototype for generated C++ functions, keys are the names of functions
- list `sparse_functions`
list of sparse functions
- list `sensi_functions`
list of sensitivity functions
- list `multiobs_functions`
list of multiobs functions
- dictionary `symbol_to_type`
indicates which type some attributes in ODEModel should have

9.2.1 Detailed Description

!/usr/bin/env python3

9.2.2 Function Documentation

9.2.2.1 var_in_function_signature()

```
def amici.ode_export.var_in_function_signature (
    name,
    varname )
```

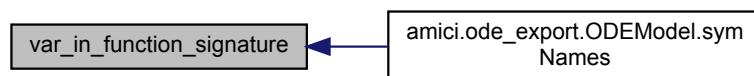
Parameters

<code>name</code>	name of the function Type: str
<code>varname</code>	name of the symbolic variable Type: str

Returns

Definition at line 277 of file `ode_export.py`.

Here is the caller graph for this function:



9.2.2.2 getSymbolicDiagonal()

```
def amici.ode_export.getSymbolicDiagonal (
    matrix )
```

Parameters

<i>matrix</i>	Matrix from which to return the diagonal Type: symengine.DenseMatrix
---------------	--

Returns

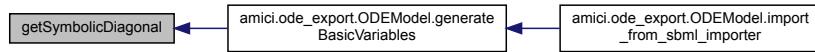
A Symbolic matrix with the diagonal of *matrix*.

Exceptions

<i>Exception</i>	The provided matrix was not square
------------------	------------------------------------

Definition at line 2235 of file `ode_export.py`.

Here is the caller graph for this function:



9.2.2.3 applyTemplate()

```
def amici.ode_export.applyTemplate (
    sourceFile,
    targetFile,
    templateData )
```

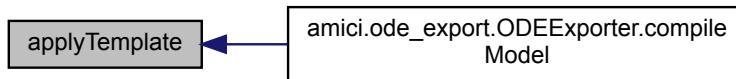
Parameters

<i>sourceFile</i>	relative or absolute path to template file Type: str
<i>targetFile</i>	relative or absolute path to output file Type: str
<i>templateData</i>	template keywords to substitute (key is template variable without <code>TemplateAmici.delimiter</code>) Type: dict

Returns

Definition at line 2271 of file `ode_export.py`.

Here is the caller graph for this function:

**9.2.2.4 sanitize_basic_sympy()**

```
def amici.ode_export.sanitize_basic_sympy ( basic )
```

Parameters

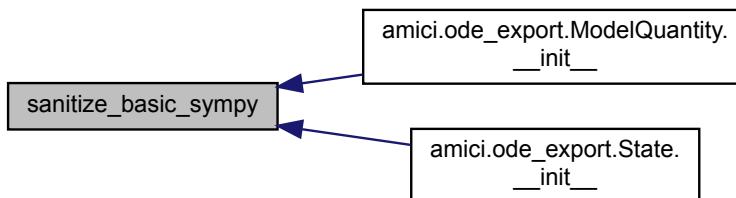
<i>basic</i>	symbolic expression
	Type: sympy.Basic

Returns

sanitized sympy.Basic

Definition at line 2290 of file `ode_export.py`.

Here is the caller graph for this function:

**9.2.3 Variable Documentation**

9.2.3.1 functions

dictionary functions

signature: defines the argument part of the function signature, input variables should have a const flag

assume_pow_positivity: identifies the functions on which assume_pow_positivity will have an effect when specified during model generation. generally these are functions that are used for solving the ODE, where negative values may negatively affect convergence of the integration algorithm

sparse: specifies whether the result of this function will be stored in sparse format. sparse format means that the function will only return an array of nonzero values and not a full matrix.

Definition at line 40 of file ode_export.py.

9.2.3.2 sparse_functions

list sparse_functions

Initial value:

```
1 = [
2     function for function in functions
3     if 'sparse' in functions[function]
4     and functions[function]['sparse']
5 ]
```

Definition at line 246 of file ode_export.py.

9.2.3.3 sensi_functions

list sensi_functions

Initial value:

```
1 = [
2     function for function in functions
3     if 'const int ip' in functions[function]['signature']
4     and function is not 'sxdot'
5 ]
```

Definition at line 252 of file ode_export.py.

9.2.3.4 multiobs_functions

list multiobs_functions

Initial value:

```
1 = [
2     function for function in functions
3     if 'const int iy' in functions[function]['signature']
4 ]
```

Definition at line 258 of file ode_export.py.

9.2.3.5 symbol_to_type

dictionary symbol_to_type

Initial value:

```

1 = {
2   'species': State,
3   'parameter': Parameter,
4   'fixed_parameter': Constant,
5   'observable': Observable,
6   'sigmay': SigmaY,
7   'llhy': LogLikelihood,
8   'expression': Expression,
9 }
```

Definition at line 549 of file `ode_export.py`.

9.3 amici.plotting Namespace Reference

Plotting related functions.

Functions

- def [plotStateTrajectories](#) (rdata, state_indices=None, ax=None)
Plot state trajectories.
- def [plotObservableTrajectories](#) (rdata, observable_indices=None, ax=None)
Plot observable trajectories.

9.3.1 Function Documentation

9.3.1.1 plotStateTrajectories()

```
def amici.plotting.plotStateTrajectories (
    rdata,
    state_indices = None,
    ax = None )
```

Parameters

<code>rdata</code>	AMICI simulation results as returned by <code>amici.getSimulationResults()</code>
<code>state_indices</code>	Indices of states for which trajectories are to be plotted
<code>ax</code>	<code>matplotlib.axes.Axes</code> instance to plot into

Returns

Definition at line 17 of file `plotting.py`.

9.3.1.2 plotObservableTrajectories()

```
def amici.plotting.plotObservableTrajectories (
    rdata,
    observable_indices = None,
    ax = None )
```

Parameters

<code>rdata</code>	AMICI simulation results as returned by <code>amici.getSimulationResults()</code>
<code>observable_indices</code>	Indices of observables for which trajectories are to be plotted
<code>ax</code>	<code>matplotlib.axes.Axes</code> instance to plot into

Returns

Definition at line 42 of file `plotting.py`.

9.4 amici.sbml_import Namespace Reference

The python sbml import module for python.

Classes

- class [SBMLError](#)
- class [SbmlImporter](#)

The `SbmlImporter` class generates AMICI C++ files for a model provided in the Systems Biology Markup Language (SBML).

Functions

- def [getRuleVars](#) (`rules`)

Extract free symbols in SBML rule formulas.
- def [assignmentRules2observables](#) (`sbml_model`, `filter_function=lambda *_:True`)

Turn assignment rules into observables.
- def [constantSpeciesToParameters](#) (`sbml_model`)

Convert constant species in the SBML model to constant parameters.
- def [replaceLogAB](#) (`x`)

Replace log(a, b) in the given string by ln(b)/ln(a)
- def [l2s](#) (`inputs`)

transforms an list into list of strings

Variables

- dictionary [default_symbols](#)

default dict for symbols

9.4.1 Detailed Description

!/usr/bin/env python3

9.4.2 Function Documentation

9.4.2.1 getRuleVars()

```
def amici.sbml_import.getRuleVars (
    rules )
```

Parameters

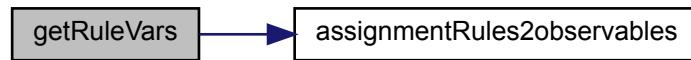
<i>rules</i>	sbml definitions of rules
	Type: list

Returns

Tuple of free symbolic variables in the formulas all provided rules

Definition at line 995 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.2 assignmentRules2observables()

```
def amici.sbml_import.assignmentRules2observables (
    sbml_model,
    filter_function = lambda *_: True )
```

Parameters

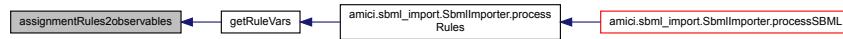
<i>sbml_model</i>	an sbml Model instance
<i>filter_function</i>	callback function taking assignment variable as input and returning True/False to indicate if the respective rule should be turned into an observable

Returns

A dictionary(observableId:{ 'name': observableNamem, 'formula': formulaString })

Definition at line 1021 of file sbml_import.py.

Here is the caller graph for this function:

**9.4.2.3 constantSpeciesToParameters()**

```
def amici.sbml_import.constantSpeciesToParameters (
    sbml_model )
```

Parameters

<i>sbml_model</i>	libsbml model instance
-------------------	------------------------

Returns

species IDs that have been turned into constants

Definition at line 1052 of file sbml_import.py.

9.4.2.4 replaceLogAB()

```
def amici.sbml_import.replaceLogAB (
    x )
```

Works for nested parentheses and nested 'log's. This can be used to circumvent the incompatible argument order between symengine (log(x, basis)) and libsbml (log(basis, x)).

Parameters

<i>x</i>	string to replace
	Type: str

Returns

string with replaced 'log's

Definition at line 1100 of file sbml_import.py.

Here is the caller graph for this function:

**9.4.2.5 l2s()**

```
def amici.sbml_import.l2s (
    inputs )
```

Parameters

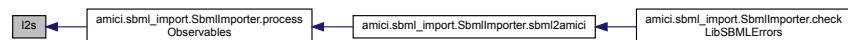
<i>inputs</i>	objects
	Type: list

Returns

list of str(object)

Definition at line 1146 of file sbml_import.py.

Here is the caller graph for this function:

**9.4.3 Variable Documentation****9.4.3.1 default_symbols**

dictionary default_symbols

Initial value:

```
1 = {
2   'species': {},
3   'parameter': {},
4   'fixed_parameter': {},
5   'observable': {},
6   'expression': {},
7   'sigmay': {},
8   'my': {},
9   'llhy': {},
10 }
```

Definition at line 17 of file sbml_import.py.

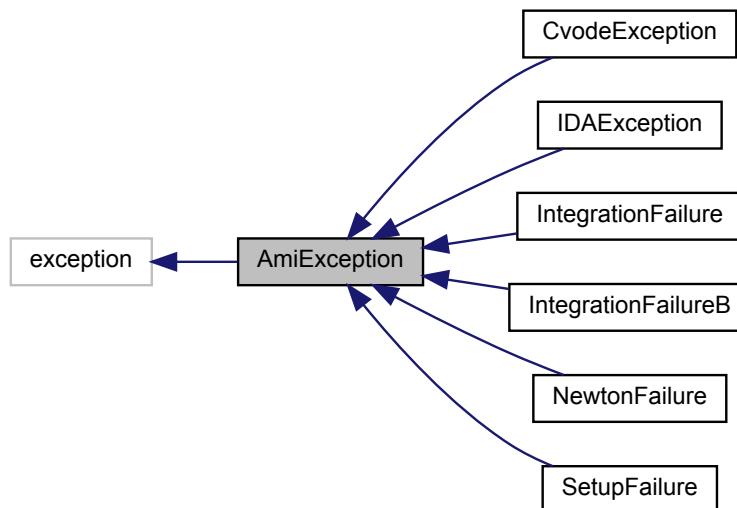
10 Class Documentation

10.1 AmiException Class Reference

amici exception handler class

```
#include <exception.h>
```

Inheritance diagram for AmiException:



Public Member Functions

- [AmiException \(char const *fmt,...\)](#)
- [AmiException \(const AmiException &old\)](#)
- [const char * what \(\) const throw \(\)](#)
- [const char * getBacktrace \(\) const](#)
- [void storeBacktrace \(const int nMaxFrames\)](#)

10.1.1 Detailed Description

has a printf style interface to allow easy generation of error messages

Definition at line 29 of file exception.h.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 AmiException() [1/2]

```
AmiException (
    char const * fmt,
    ... )
```

constructor with printf style interface

Parameters

<i>fmt</i>	error message with printf format
...	printf formating variables

Definition at line 31 of file exception.h.

Here is the call graph for this function:



10.1.2.2 AmiException() [2/2]

```
AmiException (const AmiException & old)
```

copy constructor

Parameters

<i>old</i>	object to copy from
------------	---------------------

Definition at line 43 of file exception.h.

10.1.3 Member Function Documentation

10.1.3.1 what()

```
const char* what ( ) const throw ()
```

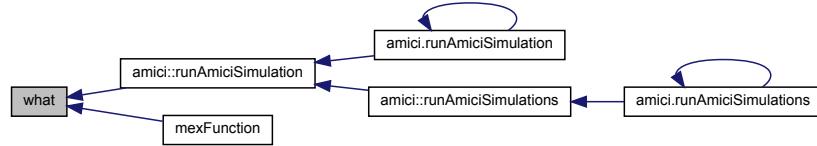
override of default error message function

Returns

```
msg error message
```

Definition at line 54 of file exception.h.

Here is the caller graph for this function:

**10.1.3.2 getBacktrace()**

```
const char* getBacktrace( ) const
```

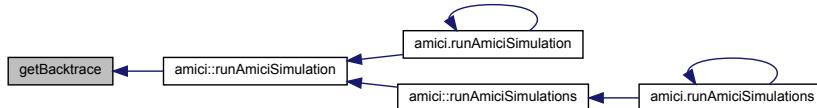
returns the stored backtrace

Returns

```
trace backtrace
```

Definition at line 61 of file exception.h.

Here is the caller graph for this function:

**10.1.3.3 storeBacktrace()**

```
void storeBacktrace(
    const int nMaxFrames )
```

stores the current backtrace

Parameters

<code>nMaxFrames</code>	number of frames to go back in stacktrace
-------------------------	---

Definition at line 68 of file exception.h.

Here is the caller graph for this function:



10.2 AmiVector Class Reference

```
#include <vector.h>
```

Public Member Functions

- `AmiVector (const long int length)`
- `AmiVector (std::vector< realtype > rvec)`
- `AmiVector (const AmiVector &vold)`
- `AmiVector & operator= (AmiVector const &other)`
- `realtype * data ()`
- `const realtype * data () const`
- `N_Vector getNVector () const`
- `std::vector< realtype > const & getVector () const`
- `int getLength () const`
- `void reset ()`
- `void minus ()`
- `void set (realtype val)`
- `realtype & operator[] (int pos)`
- `realtype & at (int pos)`

10.2.1 Detailed Description

`AmiVector` class provides a generic interface to the `NVector_Serial` struct

Definition at line 11 of file vector.h.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 AmiVector() [1/3]

```
AmiVector (
    const long int length )
```

Creates an `std::vector<realtype>` and attaches the data pointer to a newly created `N_Vector_Serial`. Using `N_VMake_Serial` ensures that the `N_Vector` module does not try to deallocate the data vector when calling `N_VDestroy_Serial`

Parameters

<i>length</i>	number of elements in vector
---------------	------------------------------

Returns

new [AmiVector](#) instance

Definition at line 23 of file vector.h.

10.2.2.2 AmiVector() [2/3]

```
AmiVector (
    std::vector< realtype > rvec )
```

constructor from std::vector, copies data from std::vector and constructs an nvec that points to the data

Parameters

<i>rvec</i>	vector from which the data will be copied
-------------	---

Returns

new [AmiVector](#) instance

Definition at line 34 of file vector.h.

10.2.2.3 AmiVector() [3/3]

```
AmiVector (
    const AmiVector & vold )
```

copy constructor

Parameters

<i>vold</i>	vector from which the data will be copied
-------------	---

Definition at line 43 of file vector.h.

10.2.3 Member Function Documentation

10.2.3.1 operator=()

```
AmiVector& operator= (
    AmiVector const & other )
```

copy-move assignment operator

Parameters

<i>other</i>	right hand side
--------------	-----------------

Returns

left hand side

Definition at line 52 of file vector.h.

10.2.3.2 data() [1/2]

```
realtyp* data ( )
```

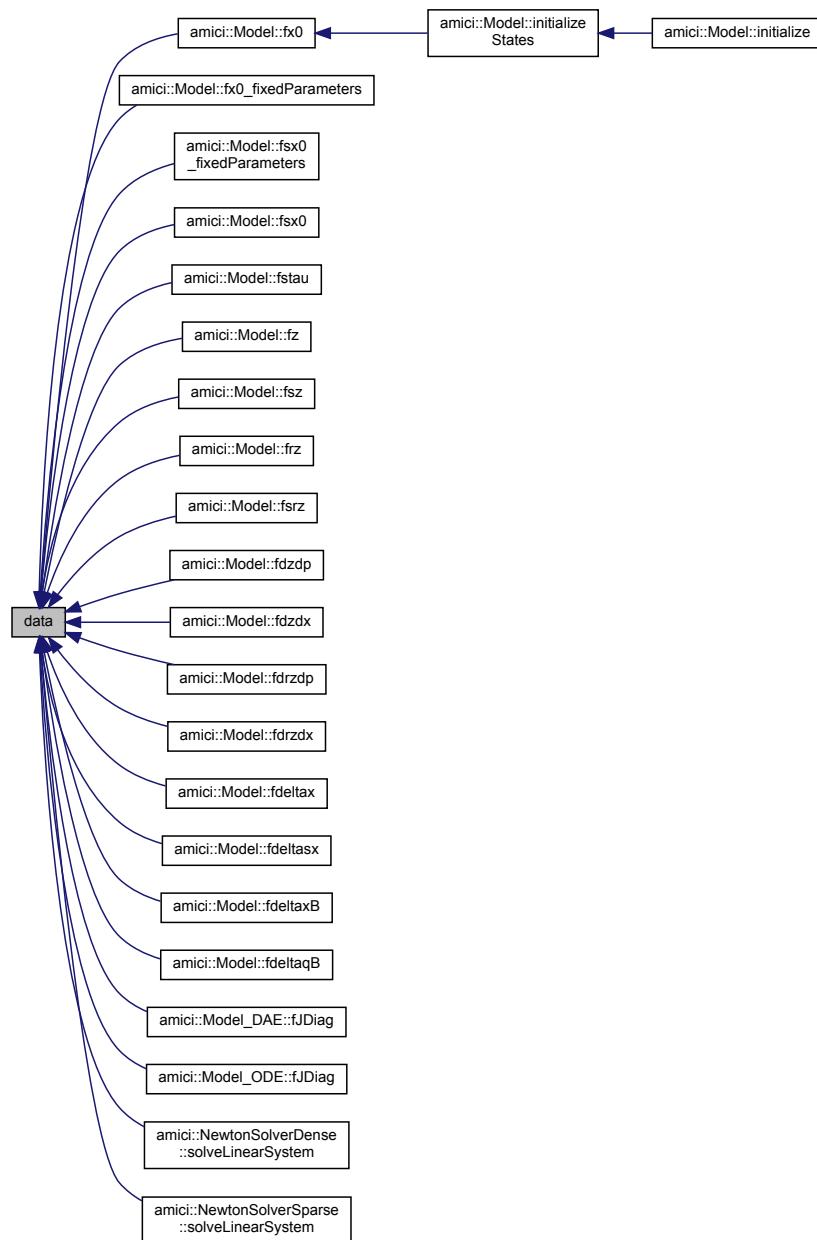
data accessor

Returns

pointer to data array

Definition at line 63 of file vector.h.

Here is the caller graph for this function:



10.2.3.3 data() [2/2]

```
const realtype* data( ) const
const data accessor
```

Returns

const pointer to data array

Definition at line 70 of file vector.h.

10.2.3.4 getNVector()

```
N_Vector getNVector ( ) const
```

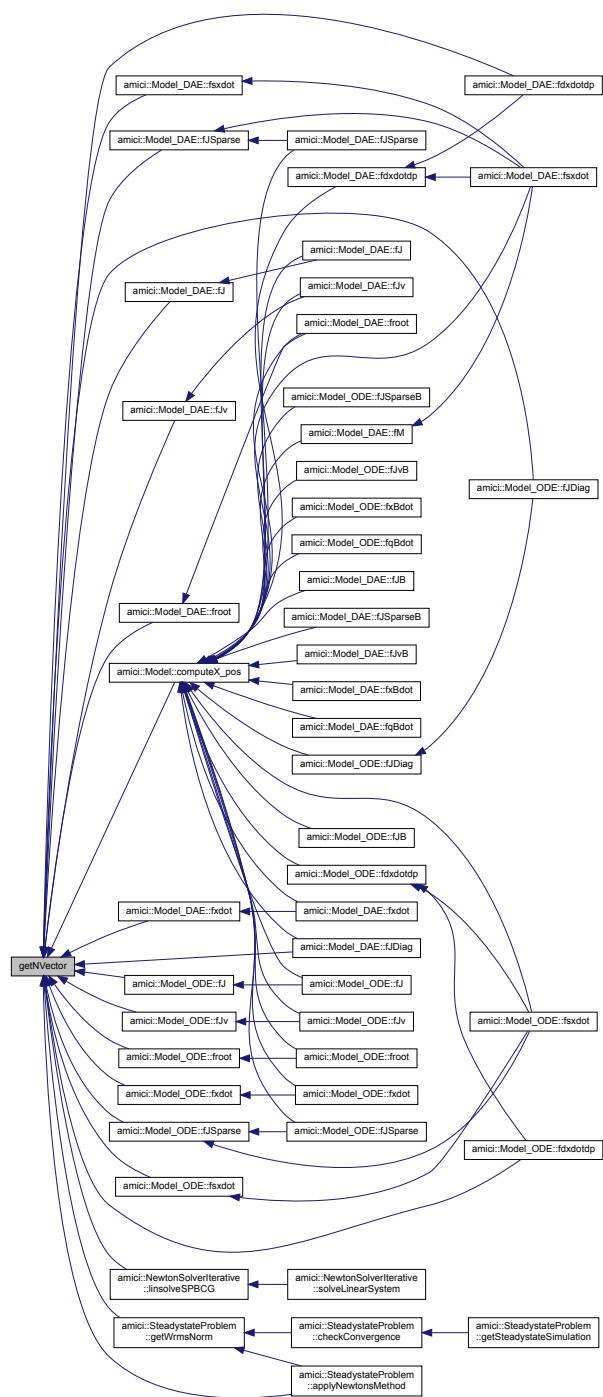
N_Vector accessor

Returns

N_Vector

Definition at line 77 of file vector.h.

Here is the caller graph for this function:



10.2.3.5 getVector()

```
std::vector<realtype> const& getVector ( ) const
```

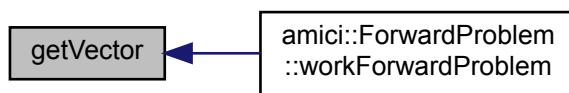
Vector accessor

Returns

Vector

Definition at line 84 of file vector.h.

Here is the caller graph for this function:



10.2.3.6 getLength()

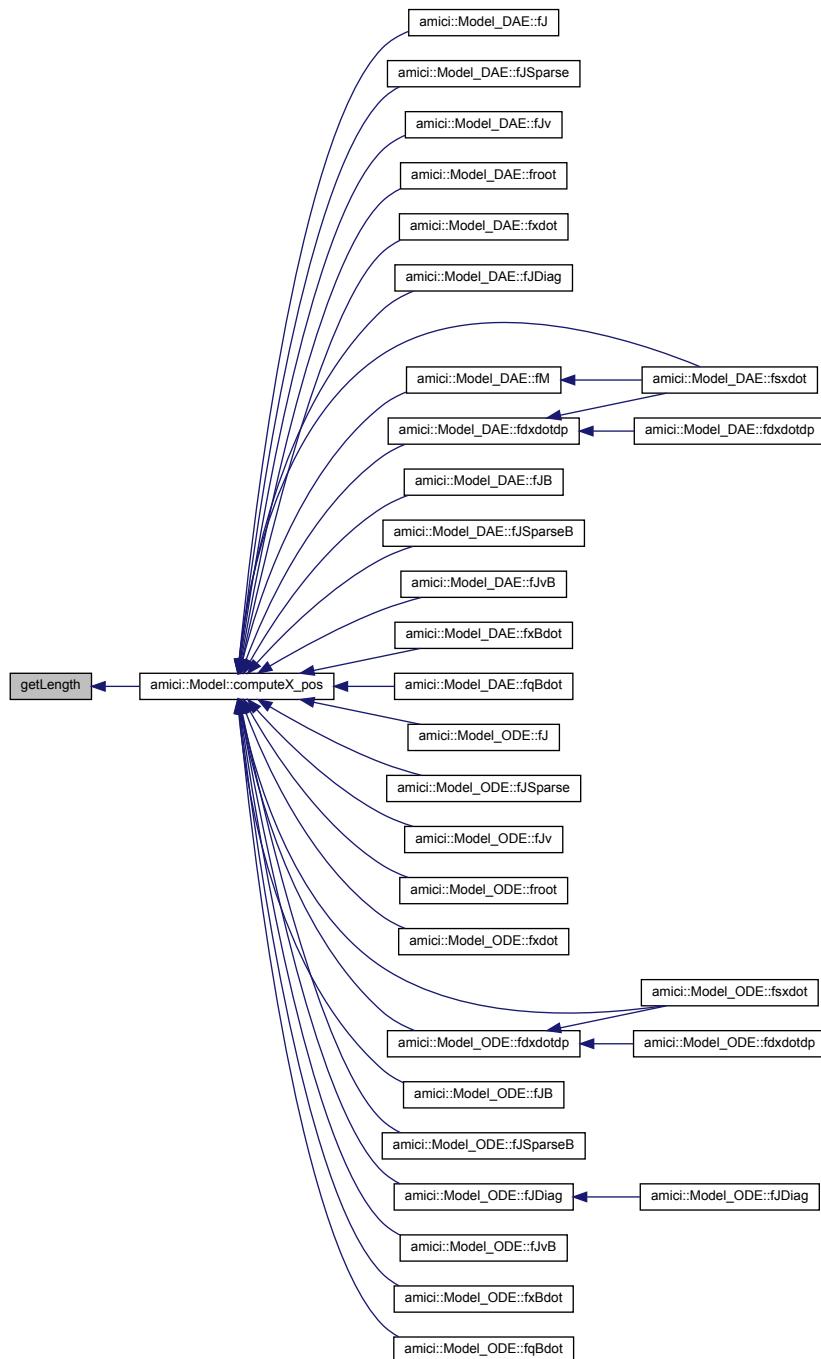
```
int getLength( ) const
```

returns the length of the vector

Returns`length`

Definition at line 91 of file vector.h.

Here is the caller graph for this function:



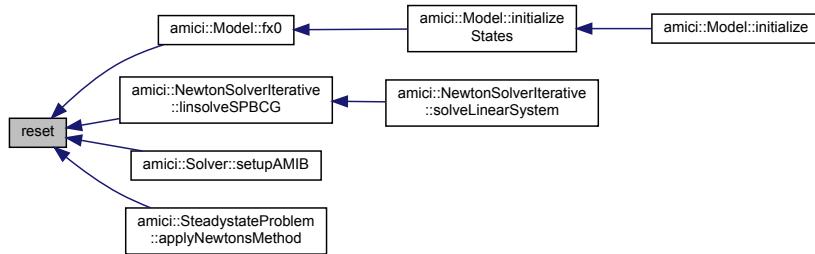
10.2.3.7 reset()

```
void reset ( )
```

resets the Vector by filling with zero values

Definition at line 97 of file vector.h.

Here is the caller graph for this function:



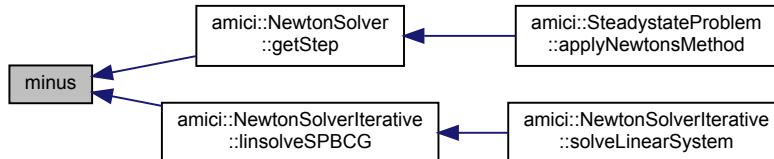
10.2.3.8 minus()

```
void minus ( )
```

changes the sign of data elements

Definition at line 103 of file vector.h.

Here is the caller graph for this function:



10.2.3.9 set()

```
void set (
    realtype val )
```

sets all data elements to a specific value

Parameters

<i>val</i>	value for data elements
------------	-------------------------

Definition at line 112 of file vector.h.

Here is the caller graph for this function:

**10.2.3.10 operator[]()**

```
realtype& operator[] (
    int pos )
```

accessor to data elements of the vector

Parameters

<i>pos</i>	index of element
------------	------------------

Returns

element

Definition at line 120 of file vector.h.

10.2.3.11 at()

```
realtype& at (
    int pos )
```

accessor to data elements of the vector

Parameters

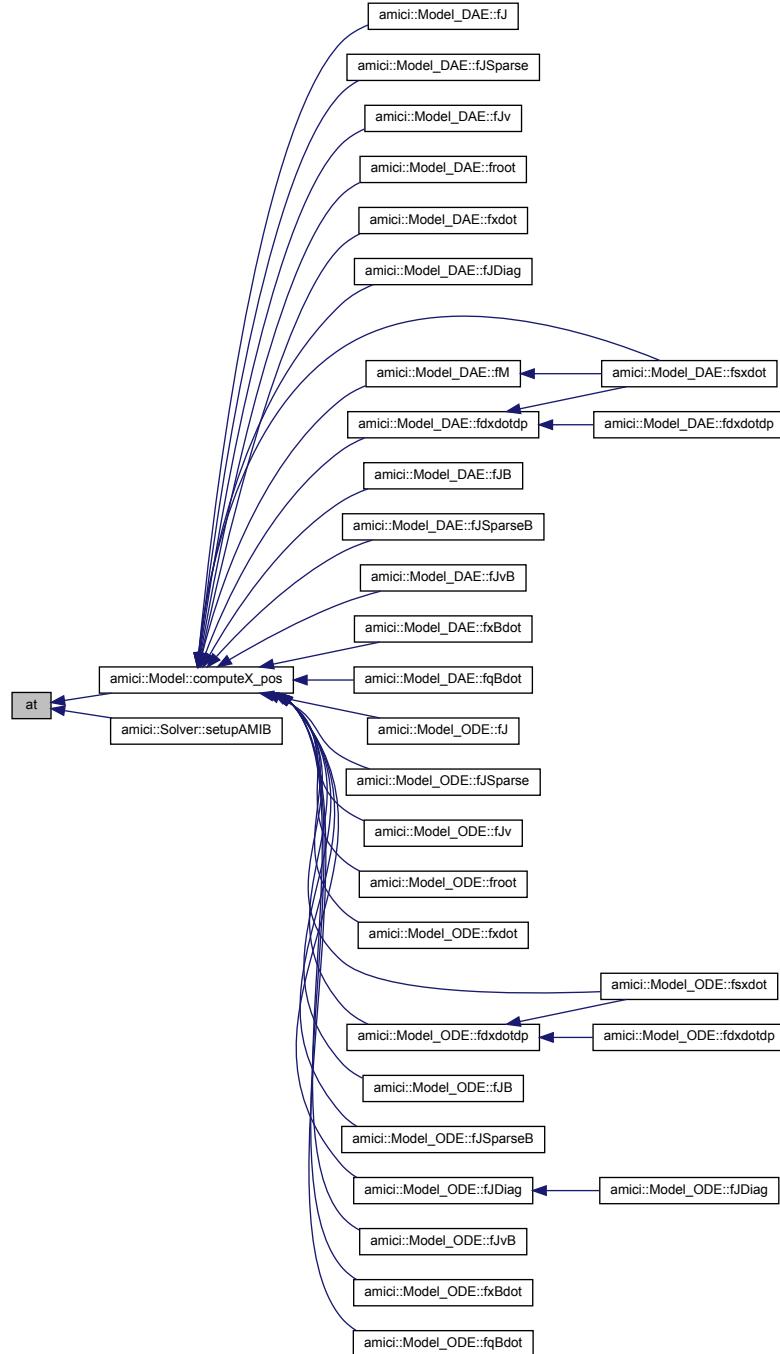
<i>pos</i>	index of element
------------	------------------

Returns

element

Definition at line 128 of file vector.h.

Here is the caller graph for this function:



10.3 AmiVectorArray Class Reference

```
#include <vector.h>
```

Public Member Functions

- [AmiVectorArray](#) (long int length_inner, long int length_outer)
- [AmiVectorArray](#) (const [AmiVectorArray](#) &vaold)
- [realtype * data](#) (int pos)
- [const realtype * data](#) (int pos) const
- [realtype & at](#) (int ipos, int jpos)
- [N_Vector * getNVectorArray](#) ()
- [N_Vector getNVector](#) (int pos)
- [AmiVector & operator\[\]](#) (int pos)
- [const AmiVector & operator\[\]](#) (int pos) const
- [int getLength](#) () const
- [void reset](#) ()

10.3.1 Detailed Description

[AmiVectorArray](#) class. provides a generic interface to arrays of NVector_Serial structs

Definition at line 148 of file vector.h.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 AmiVectorArray() [1/2]

```
AmiVectorArray (
    long int length_inner,
    long int length_outer )
```

creates an std::vector<realtype> and attaches the data pointer to a newly created N_VectorArray using Clone
 VectorArrayEmpty ensures that the N_Vector module does not try to deallocate the data vector when calling N_V
 DestroyVectorArray_Serial

Parameters

<i>length_inner</i>	length of vectors
<i>length_outer</i>	number of vectors

Returns

New [AmiVectorArray](#) instance

Definition at line 159 of file vector.h.

10.3.2.2 AmiVectorArray() [2/2]

```
AmiVectorArray (
    const AmiVectorArray & vaold )
```

copy constructor

Parameters

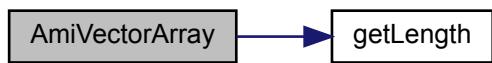
<i>vaold</i>	object to copy from
--------------	---------------------

Returns

new [AmiVectorArray](#) instance

Definition at line 173 of file vector.h.

Here is the call graph for this function:



10.3.3 Member Function Documentation

10.3.3.1 `data()` [1/2]

```
realtype* data ( int pos )
```

accessor to data of [AmiVector](#) elements

Parameters

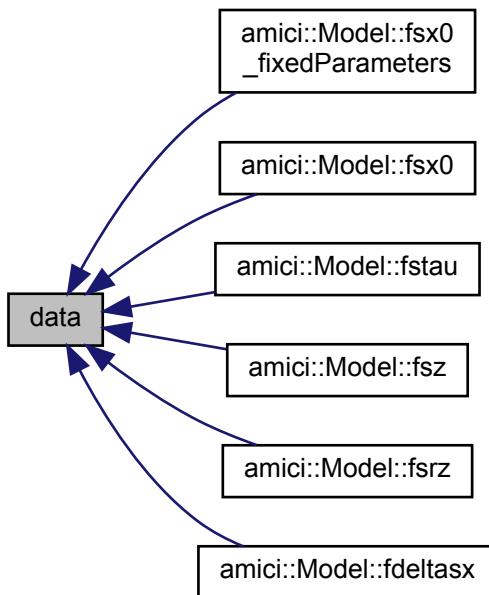
<i>pos</i>	index of AmiVector
------------	------------------------------------

Returns

pointer to data array

Definition at line 184 of file vector.h.

Here is the caller graph for this function:

**10.3.3.2 data() [2/2]**

```
const realtype* data ( int pos ) const  
const accessor to data of AmiVector elements
```

Parameters

<code>pos</code>	index of AmiVector
------------------	------------------------------------

Returns

const pointer to data array

Definition at line 192 of file vector.h.

10.3.3.3 at()

```
realtype& at (
    int ipos,
    int jpos )
```

accessor to elements of [AmiVector](#) elements

Parameters

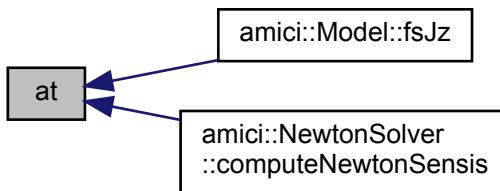
<i>ipos</i>	inner index in AmiVector
<i>jpos</i>	outer index in AmiVectorArray

Returns

element

Definition at line 201 of file `vector.h`.

Here is the caller graph for this function:



10.3.3.4 getNVectorArray()

```
N_Vector* getNVectorArray ( )
```

accessor to NVectorArray

Returns

`N_VectorArray`

Definition at line 208 of file `vector.h`.

10.3.3.5 getNVector()

```
N_Vector getNVector (
    int pos )
```

accessor to NVector element

Parameters

<i>pos</i>	index of corresponding AmiVector
------------	--

Returns[N_Vector](#)

Definition at line 216 of file vector.h.

10.3.3.6 operator[]() [1/2]

```
AmiVector& operator[] ( int pos )
```

accessor to [AmiVector](#) elements

Parameters

<i>pos</i>	index of AmiVector
------------	------------------------------------

Returns[AmiVector](#)

Definition at line 224 of file vector.h.

10.3.3.7 operator[]() [2/2]

```
const AmiVector& operator[] ( int pos ) const
```

const accessor to [AmiVector](#) elements

Parameters

<i>pos</i>	index of AmiVector
------------	------------------------------------

Returns[const AmiVector](#)

Definition at line 232 of file vector.h.

10.3.3.8 `getLength()`

`int getLength () const`

length of [AmiVectorArray](#)

Returns

`length`

Definition at line 239 of file `vector.h`.

Here is the caller graph for this function:



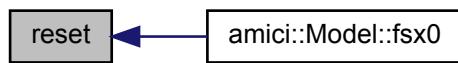
10.3.3.9 `reset()`

`void reset ()`

resets every [AmiVector](#) in [AmiVectorArray](#)

Definition at line 244 of file `vector.h`.

Here is the caller graph for this function:



10.4 BackwardProblem Class Reference

class to solve backwards problems.

```
#include <backwardproblem.h>
```

Public Member Functions

- void [workBackwardProblem \(\)](#)
- [BackwardProblem \(const ForwardProblem *fwd\)](#)
- [realtype gett \(\) const](#)
- [int getwhich \(\) const](#)
- [int * getwhichptr \(\)](#)
- [AmiVector * getxBptr \(\)](#)
- [AmiVector * getxQBptr \(\)](#)
- [AmiVector * getdxBptr \(\)](#)
- [std::vector< realtype > const & getdJydx \(\) const](#)

10.4.1 Detailed Description

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

Definition at line 23 of file backwardproblem.h.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 BackwardProblem()

```
BackwardProblem (
    const ForwardProblem * fwd )
```

Construct backward problem from forward problem

Parameters

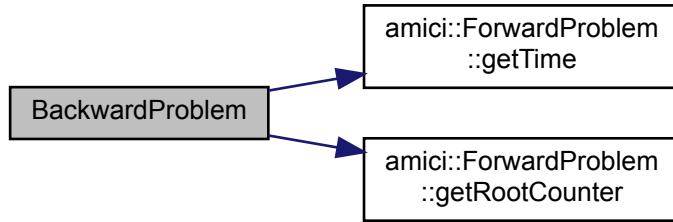
<i>fwd</i>	pointer to corresponding forward problem
------------	--

Returns

new [BackwardProblem](#) instance

Definition at line 18 of file backwardproblem.cpp.

Here is the call graph for this function:



10.4.3 Member Function Documentation

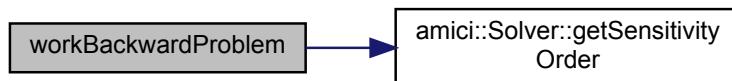
10.4.3.1 workBackwardProblem()

```
void workBackwardProblem ( )
```

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

Definition at line 42 of file backwardproblem.cpp.

Here is the call graph for this function:



10.4.3.2 gett()

```
realtype gett ( ) const
```

accessor for t

Returns

t

Definition at line 32 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.3 getwhich()

```
int getwhich ( ) const
```

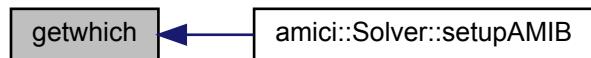
accessor for which

Returns

which

Definition at line 39 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.4 getwhichptr()

```
int* getwhichptr ( )
```

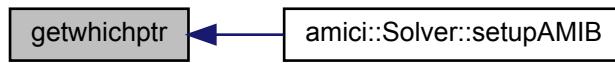
accessor for pointer to which

Returns

which

Definition at line 46 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.5 getxBptr()

```
AmiVector* getxBptr ( )
```

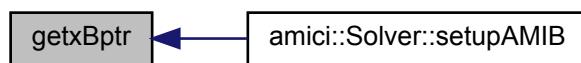
accessor for pointer to xB

Returns

&x_B

Definition at line 53 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.6 getxQBptr()

```
AmiVector* getxQBptr ( )
```

accessor for pointer to xQB

Returns

&xQB

Definition at line 60 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.7 getdxBptr()

```
AmiVector* getdxBptr ( )
```

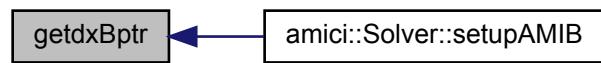
accessor for pointer to dxB

Returns

&dxB

Definition at line 67 of file backwardproblem.h.

Here is the caller graph for this function:



10.4.3.8 `getdJydx()`

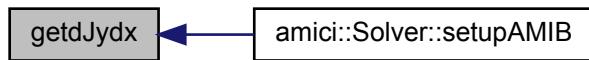
```
std::vector<realtype> const& getdJydx ( ) const
accessor for dJydx
```

Returns

`dJydx`

Definition at line 74 of file `backwardproblem.h`.

Here is the caller graph for this function:

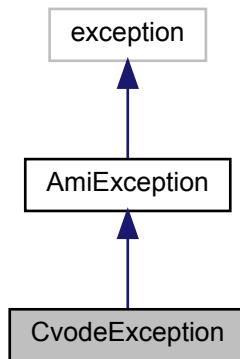


10.5 CvodeException Class Reference

cicode exception handler class

```
#include <exception.h>
```

Inheritance diagram for CvodeException:



Public Member Functions

- `CvodeException (const int error_code, const char *function)`

10.5.1 Detailed Description

Definition at line 117 of file exception.h.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 CvodeException()

```
CvodeException (
    const int error_code,
    const char * function )
```

constructor

Parameters

<i>error_code</i>	error code returned by cvode function
<i>function</i>	cvode function name

Definition at line 123 of file exception.h.

10.6 ExpData Class Reference

[ExpData](#) carries all information about experimental or condition-specific data.

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

Public Member Functions

- [ExpData \(\)](#)
- [ExpData \(const ExpData &\)=default](#)
- [ExpData \(int nytrue, int nztrue, int nmaxevent\)](#)
- [ExpData \(int nytrue, int nztrue, int nmaxevent, std::vector< realtype > ts\)](#)
- [ExpData \(int nytrue, int nztrue, int nmaxevent, std::vector< realtype > ts, std::vector< realtype > observedData, std::vector< realtype > observedDataStdDev, std::vector< realtype > observedEvents, std::vector< realtype > observedEventsStdDev\)](#)
- [ExpData \(const Model &model\)](#)
- [ExpData \(const ReturnData &rdata, realtype sigma_y, realtype sigma_z\)](#)
- [ExpData \(const ReturnData &rdata, std::vector< realtype > sigma_y, std::vector< realtype > sigma_z\)](#)
- [int nytrue \(\) const](#)
- [int nztrue \(\) const](#)
- [int nmaxevent \(\) const](#)
- [int nt \(\) const](#)
- [void setTimepoints \(const std::vector< realtype > &ts\)](#)
- [std::vector< realtype > const & getTimepoints \(\) const](#)
- [realtype getTimepoint \(int it\) const](#)
- [void setObservedData \(const std::vector< realtype > &observedData\)](#)
- [void setObservedData \(const std::vector< realtype > &observedData, int iy\)](#)

- bool `isSetObservedData` (int it, int iy) const
- std::vector< `realtype` > const & `getObservedData` () const
- const `realtype` * `getObservedDataPtr` (int it) const
- void `setObservedDataStdDev` (const std::vector< `realtype` > &`observedDataStdDev`)
- void `setObservedDataStdDev` (const `realtype` stdDev)
- void `setObservedDataStdDev` (const std::vector< `realtype` > &`observedDataStdDev`, int iy)
- void `setObservedDataStdDev` (const `realtype` stdDev, int iy)
- bool `isSetObservedDataStdDev` (int it, int iy) const
- std::vector< `realtype` > const & `getObservedDataStdDev` () const
- const `realtype` * `getObservedDataStdDevPtr` (int it) const
- void `setObservedEvents` (const std::vector< `realtype` > &`observedEvents`)
- void `setObservedEvents` (const std::vector< `realtype` > &`observedEvents`, int iz)
- bool `isSetObservedEvents` (int ie, int iz) const
- std::vector< `realtype` > const & `getObservedEvents` () const
- const `realtype` * `getObservedEventsPtr` (int ie) const
- void `setObservedEventsStdDev` (const std::vector< `realtype` > &`observedEventsStdDev`)
- void `setObservedEventsStdDev` (const `realtype` stdDev)
- void `setObservedEventsStdDev` (const std::vector< `realtype` > &`observedEventsStdDev`, int iz)
- void `setObservedEventsStdDev` (const `realtype` stdDev, int iz)
- bool `isSetObservedEventsStdDev` (int ie, int iz) const
- std::vector< `realtype` > const & `getObservedEventsStdDev` () const
- const `realtype` * `getObservedEventsStdDevPtr` (int ie) const

Public Attributes

- std::vector< `realtype` > `fixedParameters`
- std::vector< `realtype` > `fixedParametersPreequilibration`
- std::vector< `realtype` > `fixedParametersPresimulation`
- `realtype t_presim = 0`

duration of pre-simulation if this is > 0, presimulation will be performed from (model->t0 - t_presim) to model->t0 using the fixedParameters in fixedParametersPresimulation

Protected Member Functions

- void `checkDataDimension` (std::vector< `realtype` > input, const char *fieldname) const
- void `checkEventsDimension` (std::vector< `realtype` > input, const char *fieldname) const
- void `checkSigmaPositivity` (std::vector< `realtype` > sigmaVector, const char *vectorName) const
- void `checkSigmaPositivity` (`realtype` sigma, const char *sigmaName) const

Protected Attributes

- int `nytrue_`
- int `nztrue_`
- int `nmaxevent_`
- std::vector< `realtype` > `ts`
- std::vector< `realtype` > `observedData`
- std::vector< `realtype` > `observedDataStdDev`
- std::vector< `realtype` > `observedEvents`
- std::vector< `realtype` > `observedEventsStdDev`

10.6.1 Detailed Description

Definition at line 14 of file edata.h.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 ExpData() [1/8]

```
ExpData ( )
```

default constructor

Definition at line 14 of file edata.cpp.

10.6.2.2 ExpData() [2/8]

```
ExpData ( const ExpData & ) [default]
```

default copy constructor, needs to be declared to be generated in swig

10.6.2.3 ExpData() [3/8]

```
ExpData ( int nytrue, int nztrue, int nmaxevent )
```

constructor that only initializes dimensions

Parameters

<i>nytrue</i>	
<i>nztrue</i>	
<i>nmaxevent</i>	

Definition at line 16 of file edata.cpp.

10.6.2.4 ExpData() [4/8]

```
ExpData ( int nytrue, int nztrue, int nmaxevent, std::vector< realltype > ts )
```

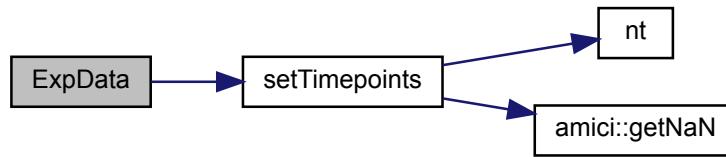
constructor that initializes timepoints from vectors

Parameters

<i>nytrue</i>	(dimension: scalar)
<i>nztrue</i>	(dimension: scalar)
<i>nmaxevent</i>	(dimension: scalar)
<i>ts</i>	(dimension: nt)

Definition at line 21 of file edata.cpp.

Here is the call graph for this function:

**10.6.2.5 ExpData() [5/8]**

```
ExpData (
    int nytrue,
    int nztrue,
    int nmaxevent,
    std::vector< realltype > ts,
    std::vector< realltype > observedData,
    std::vector< realltype > observedDataStdDev,
    std::vector< realltype > observedEvents,
    std::vector< realltype > observedEventsStdDev )
```

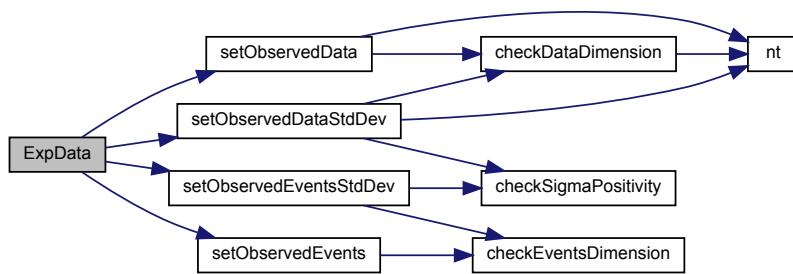
constructor that initializes timepoints and data from vectors

Parameters

<i>nytrue</i>	(dimension: scalar)
<i>nztrue</i>	(dimension: scalar)
<i>nmaxevent</i>	(dimension: scalar)
<i>ts</i>	(dimension: nt)
<i>observedData</i>	(dimension: nt x nytrue, row-major)
<i>observedDataStdDev</i>	(dimension: nt x nytrue, row-major)
<i>observedEvents</i>	(dimension: nmaxevent x nztrue, row-major)
<i>observedEventsStdDev</i>	(dimension: nmaxevent x nztrue, row-major)

Definition at line 28 of file edata.cpp.

Here is the call graph for this function:



10.6.2.6 ExpData() [6/8]

```
ExpData (
    const Model & model )
```

constructor that initializes with [Model](#)

Parameters

<i>model</i>	pointer to model specification object
--------------	---------------------------------------

Definition at line 42 of file eedata.cpp.

10.6.2.7 ExpData() [7/8]

```
ExpData (
    const ReturnData & rdata,
    realtype sigma_y,
    realtype sigma_z )
```

constructor that initializes with returnData, adds

Parameters

<i>rdata</i>	return data pointer with stored simulation results
<i>sigma_y</i>	scalar standard deviations for all observables
<i>sigma_z</i>	scalar standard deviations for all event observables

Definition at line 48 of file eedata.cpp.

10.6.2.8 ExpData() [8/8]

```
ExpData (
    const ReturnData & rdata,
    std::vector< realtype > sigma_y,
    std::vector< realtype > sigma_z )
```

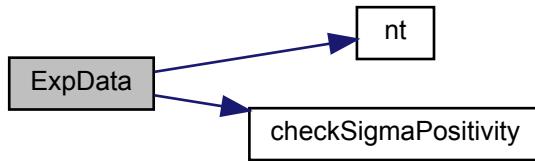
constructor that initializes with returnData, adds

Parameters

<i>rdata</i>	return data pointer with stored simulation results
<i>sigma_y</i>	vector of standard deviations for observables (dimension: nytrue or nt x nytrue, row-major)
<i>sigma_z</i>	vector of standard deviations for event observables (dimension: nztrue or nmaxevent x nztrue, row-major)

Definition at line 53 of file edata.cpp.

Here is the call graph for this function:



10.6.3 Member Function Documentation

10.6.3.1 nytrue()

```
int nytrue ( ) const
```

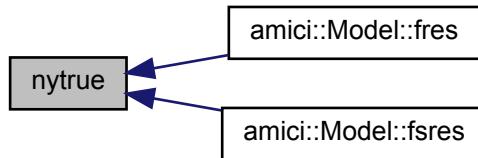
number of observables of the non-augmented model

Returns

number of observables of the non-augmented model

Definition at line 290 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.2 nztrue()**

```
int nztrue ( ) const
```

number of event observables of the non-augmented model

Returns

number of event observables of the non-augmented model

Definition at line 295 of file edata.cpp.

10.6.3.3 nmaxevent()

```
int nmaxevent ( ) const
```

maximal number of events to track

Returns

maximal number of events to track

Definition at line 300 of file edata.cpp.

10.6.3.4 nt()

```
int nt ( ) const
```

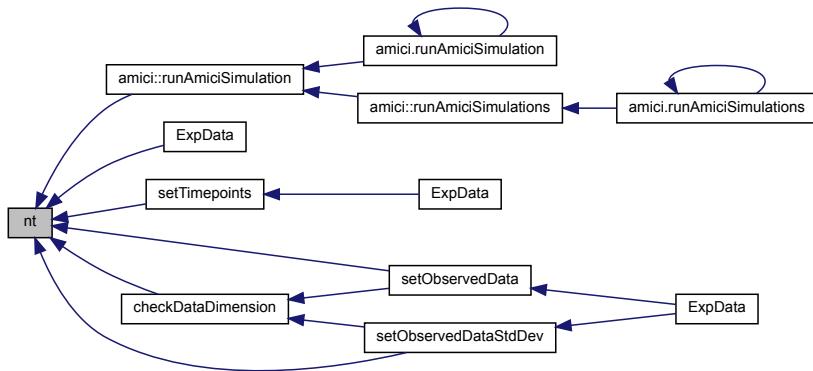
number of timepoints

Returns

number of timepoints

Definition at line 102 of file edata.cpp.

Here is the caller graph for this function:



10.6.3.5 setTimepoints()

```
void setTimepoints (
    const std::vector< realtype > & ts )
```

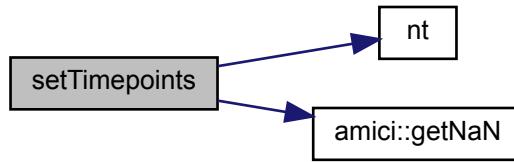
set function that copies data from input to [ExpData::ts](#)

Parameters

<i>ts</i>	timepoints
-----------	------------

Definition at line 89 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.6 getTimepoints()

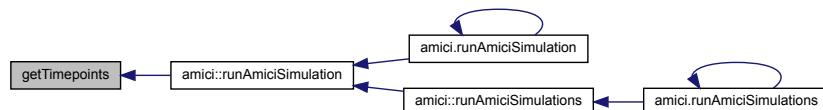
`std::vector< realtype > const & getTimepoints() const`
 get function that copies data from `ExpData::ts` to output

Returns

`ExpData::ts`

Definition at line 98 of file edata.cpp.

Here is the caller graph for this function:



10.6.3.7 getTimepoint()

`realtype getTimepoint(int it) const`
 get function that returns timepoint at index

Parameters

<i>it</i>	timepoint index
-----------	-----------------

Returns

timepoint timepoint at index

Definition at line 106 of file eedata.cpp.

10.6.3.8 setObservedData() [1/2]

```
void setObservedData (
    const std::vector< realtype > & observedData )
```

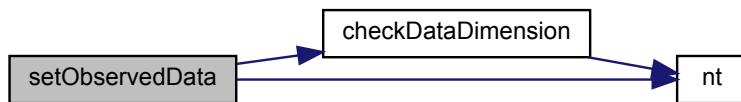
set function that copies data from input to ExpData::my

Parameters

<i>observedData</i>	observed data (dimension: nt x nytrue, row-major)
---------------------	---

Definition at line 110 of file eedata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.9 setObservedData() [2/2]

```
void setObservedData (
    const std::vector< realtypes > & observedData,
    int iy )
```

set function that copies observed data for specific observable

Parameters

<i>observedData</i>	observed data (dimension: nt)
<i>iy</i>	oberved data index

Definition at line 119 of file edata.cpp.

Here is the call graph for this function:

**10.6.3.10 isSetObservedData()**

```
bool isSetObservedData (
    int it,
    int iy ) const
```

get function that checks whether data at specified indices has been set

Parameters

<i>it</i>	time index
<i>iy</i>	observable index

Returns

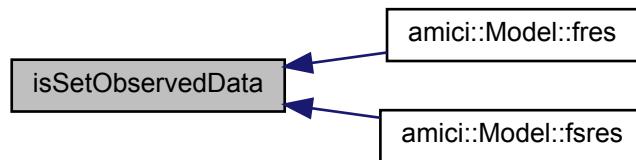
boolean specifying if data was set

Definition at line 127 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.11 getObservedData()

`std::vector< realtype > const & getObservedData() const`

get function that copies data from `ExpData::observedData` to output

Returns

observed data (dimension: nt x nytrue, row-major)

Definition at line 131 of file eedata.cpp.

10.6.3.12 getObservedDataPtr()

`const realtype * getObservedDataPtr(int it) const`

get function that returns a pointer to observed data at index

Parameters

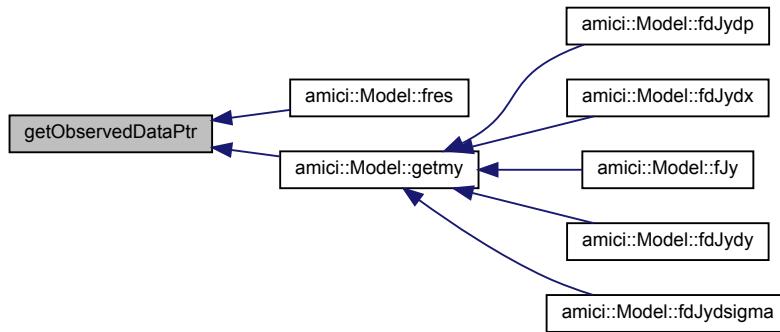
<code>it</code>	timepoint index
-----------------	-----------------

Returns

pointer to observed data at index (dimension: nytrue)

Definition at line 135 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.13 setObservedDataStdDev() [1/4]**

```
void setObservedDataStdDev (
    const std::vector< realtype > & observedDataStdDev )
```

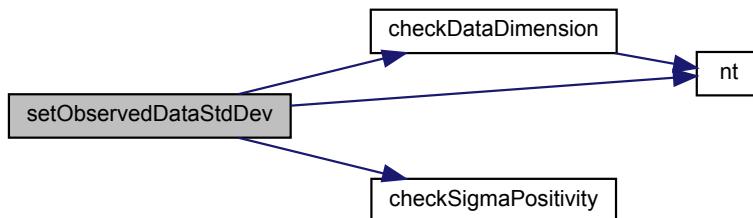
set function that copies data from input to [ExpData::observedDataStdDev](#)

Parameters

<i>observedDataStdDev</i>	standard deviation of observed data (dimension: nt x nytrue, row-major)
---------------------------	---

Definition at line 142 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.14 setObservedDataStdDev() [2/4]

```
void setObservedDataStdDev (
    const realtype stdDev )
```

set function that sets all `ExpData::observedDataStdDev` to the input value

Parameters

<code>stdDev</code>	standard deviation (dimension: scalar)
---------------------	--

Definition at line 152 of file edata.cpp.

Here is the call graph for this function:



10.6.3.15 setObservedDataStdDev() [3/4]

```
void setObservedDataStdDev (
    const std::vector< realtype > & observedDataStdDev,
    int iy )
```

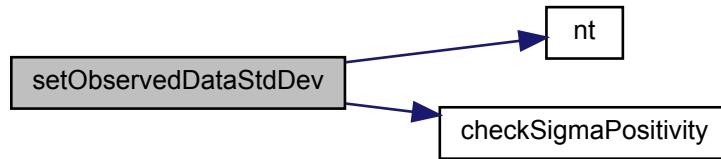
set function that copies standard deviation of observed data for specific observable

Parameters

<code>observedDataStdDev</code>	standard deviation of observed data (dimension: nt)
<code>iy</code>	observed data index

Definition at line 157 of file edata.cpp.

Here is the call graph for this function:



10.6.3.16 setObservedDataStdDev() [4/4]

```
void setObservedDataStdDev (
    const realtype stdDev,
    int iy )
```

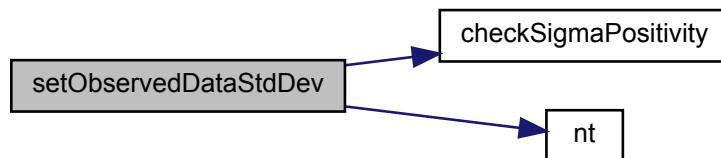
set function that sets all standard deviation of a specific observable to the input value

Parameters

<i>stdDev</i>	standard deviation (dimension: scalar)
<i>iy</i>	observed data index

Definition at line 166 of file edata.cpp.

Here is the call graph for this function:



10.6.3.17 isSetObservedDataStdDev()

```
bool isSetObservedDataStdDev (
    int it,
    int iy ) const
```

get function that checks whether standard deviation of data at specified indices has been set

Parameters

<i>it</i>	time index
<i>iy</i>	observable index

Returns

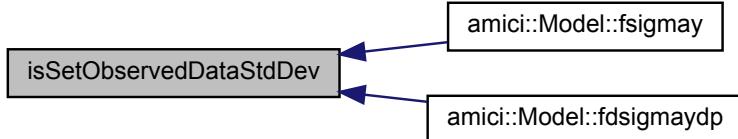
boolean specifying if standard deviation of data was set

Definition at line 172 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.18 getObservedDataStdDev()

```
std::vector< realtype > const & getObservedDataStdDev ( ) const
```

get function that copies data from `ExpData::observedDataStdDev` to output

Returns

standard deviation of observed data

Definition at line 176 of file edata.cpp.

10.6.3.19 getObservedDataStdDevPtr()

```
const realtype * getObservedDataStdDevPtr ( int it ) const
```

get function that returns a pointer to standard deviation of observed data at index

Parameters

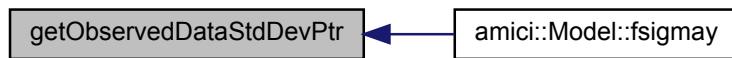
<i>it</i>	timepoint index
-----------	-----------------

Returns

pointer to standard deviation of observed data at index

Definition at line 180 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.20 setObservedEvents() [1/2]**

```
void setObservedEvents ( const std::vector< realtype > & observedEvents )
```

set function that copies observed event data from input to [ExpData::observedEvents](#)

Parameters

<i>observedEvents</i>	observed data (dimension: nmaxevent x nztrue, row-major)
-----------------------	--

Definition at line 187 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.21 setObservedEvents() [2/2]

```
void setObservedEvents (
    const std::vector< realtyp > & observedEvents,
    int iz )
```

set function that copies observed event data for specific event observable

Parameters

<i>observedEvents</i>	observed data (dimension: nmaxevent)
<i>iz</i>	observed event data index

Definition at line 196 of file edata.cpp.

10.6.3.22 isSetObservedEvents()

```
bool isSetObservedEvents (
    int ie,
    int iz ) const
```

get function that checks whether event data at specified indices has been set

Parameters

<i>ie</i>	event index
<i>iz</i>	event observable index

Returns

boolean specifying if data was set

Definition at line 205 of file edata.cpp.

Here is the call graph for this function:

**10.6.3.23 getObservedEvents()**

```
std::vector< realtype > const & getObservedEvents() const
```

get function that copies data from ExpData::mz to output

Returns

observed event data

Definition at line 209 of file edata.cpp.

10.6.3.24 getObservedEventsPtr()

```
const realtype * getObservedEventsPtr( int ie ) const
```

get function that returns a pointer to observed data at ieth occurrence

Parameters

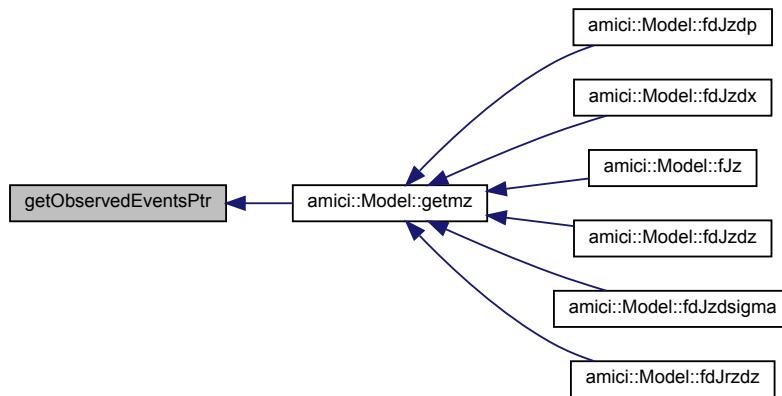
<i>ie</i>	event occurence
-----------	-----------------

Returns

pointer to observed event data at ieth occurrence

Definition at line 213 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.25 setObservedEventsStdDev() [1/4]**

```
void setObservedEventsStdDev (
    const std::vector< realtype > & observedEventsStdDev )
```

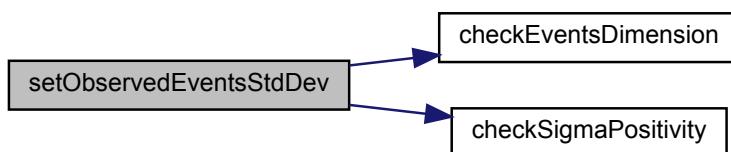
set function that copies data from input to [ExpData::observedEventsStdDev](#)

Parameters

<i>observedEventsStdDev</i>	standard deviation of observed event data
-----------------------------	---

Definition at line 220 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.26 setObservedEventsStdDev() [2/4]

```
void setObservedEventsStdDev (
    const realtype stdDev )
```

set function that sets all `ExpData::observedDataStdDev` to the input value

Parameters

<code>stdDev</code>	standard deviation (dimension: scalar)
---------------------	--

Definition at line 230 of file edata.cpp.

Here is the call graph for this function:



10.6.3.27 setObservedEventsStdDev() [3/4]

```
void setObservedEventsStdDev (
    const std::vector< realtype > & observedEventsStdDev,
    int iz )
```

set function that copies standard deviation of observed data for specific observable

Parameters

<code>observedEventsStdDev</code>	standard deviation of observed data (dimension: nmaxevent)
<code>iz</code>	observed data index

Definition at line 235 of file edata.cpp.

Here is the call graph for this function:



10.6.3.28 setObservedEventsStdDev() [4/4]

```
void setObservedEventsStdDev (
    const realtype stdDev,
    int iz )
```

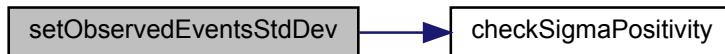
set function that sets all standard deviation of a specific observable to the input value

Parameters

<i>stdDev</i>	standard deviation (dimension: scalar)
<i>iz</i>	observed data index

Definition at line 244 of file edata.cpp.

Here is the call graph for this function:



10.6.3.29 isSetObservedEventsStdDev()

```
bool isSetObservedEventsStdDev (
    int ie,
    int iz ) const
```

get function that checks whether standard deviation of even data at specified indices has been set

Parameters

<i>ie</i>	event index
<i>iz</i>	event observable index

Returns

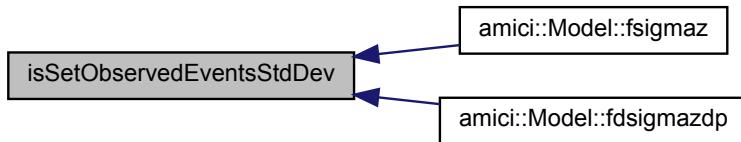
boolean specifying if standard deviation of event data was set

Definition at line 251 of file eedata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.6.3.30 getObservedEventsStdDev()**

```
std::vector< realtype > const & getObservedEventsStdDev( ) const
```

get function that copies data from [ExpData::observedEventsStdDev](#) to output

Returns

standard deviation of observed event data

Definition at line 258 of file eedata.cpp.

10.6.3.31 getObservedEventsStdDevPtr()

```
const realtype * getObservedEventsStdDevPtr(
    int ie) const
```

get function that returns a pointer to standard deviation of observed event data at ieth occurrence

Parameters

<i>ie</i>	event occurrence
-----------	------------------

Returns

pointer to standard deviation of observed event data at ieth occurrence

Definition at line 262 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.32 checkDataDimension()**

```
void checkDataDimension (
    std::vector< realltype > input,
    const char * fieldname ) const [protected]
```

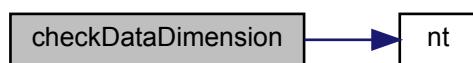
checker for dimensions of input observedData or observedDataStdDev

Parameters

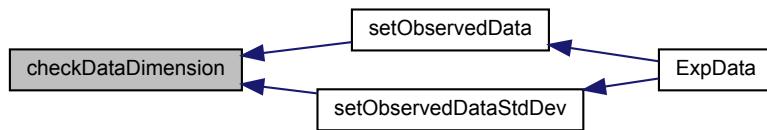
<i>input</i>	vector input to be checked
<i>fieldname</i>	name of the input

Definition at line 269 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.6.3.33 checkEventsDimension()

```
void checkEventsDimension (
    std::vector< realtype > input,
    const char * fieldname ) const [protected]
```

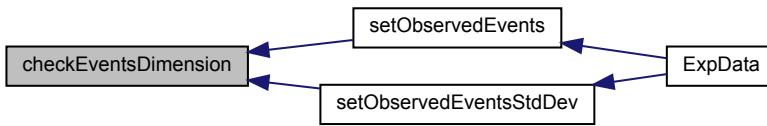
checker for dimensions of input observedEvents or observedEventsStdDev

Parameters

<i>input</i>	vector input to be checked
<i>fieldname</i>	name of the input

Definition at line 274 of file edata.cpp.

Here is the caller graph for this function:



10.6.3.34 checkSigmaPositivity() [1/2]

```
void checkSigmaPositivity (
    std::vector< realtype > sigmaVector,
    const char * vectorName ) const [protected]
```

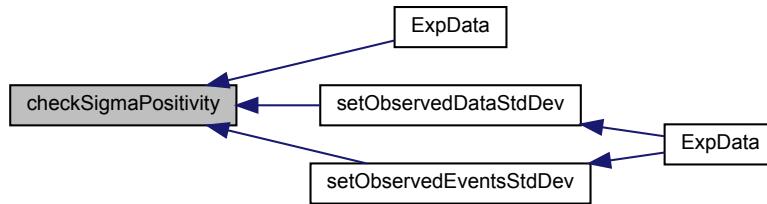
checks input vector of sigmas for not strictly positive values

Parameters

<i>sigmaVector</i>	vector input to be checked
<i>vectorName</i>	name of the input

Definition at line 279 of file edata.cpp.

Here is the caller graph for this function:

**10.6.3.35 checkSigmaPositivity() [2/2]**

```
void checkSigmaPositivity (
    realtype sigma,
    const char * sigmaName ) const [protected]
```

checks input scalar sigma for not strictly positive value

Parameters

<i>sigma</i>	input to be checked
<i>sigmaName</i>	name of the input

Definition at line 284 of file edata.cpp.

10.6.4 Member Data Documentation**10.6.4.1 fixedParameters**

```
std::vector<realtype> fixedParameters
```

condition-specific parameters of size [Model::nk\(\)](#) or empty

Definition at line 327 of file edata.h.

10.6.4.2 fixedParametersPreequilibration

```
std::vector<realtype> fixedParametersPreequilibration
```

condition-specific parameters for pre-equilibration of size [Model::nk\(\)](#) or empty. Overrides Solver::newton_preq

Definition at line 330 of file edata.h.

10.6.4.3 fixedParametersPresimulation

```
std::vector<realtype> fixedParametersPresimulation
```

condition-specific parameters for pre-simulation of size [Model::nk\(\)](#) or empty.

Definition at line 332 of file edata.h.

10.6.4.4 nytrue_

```
int nytrue_ [protected]
```

number of observables

Definition at line 375 of file edata.h.

10.6.4.5 nztrue_

```
int nztrue_ [protected]
```

number of event observables

Definition at line 378 of file edata.h.

10.6.4.6 nmaxevent_

```
int nmaxevent_ [protected]
```

maximal number of event occurrences

Definition at line 381 of file edata.h.

10.6.4.7 ts

std::vector<[realtype](#)> ts [protected]

observation timepoints (dimension: nt)

Definition at line 384 of file [edata.h](#).

10.6.4.8 observedData

std::vector<[realtype](#)> observedData [protected]

observed data (dimension: nt x nytrue, row-major)

Definition at line 387 of file [edata.h](#).

10.6.4.9 observedDataStdDev

std::vector<[realtype](#)> observedDataStdDev [protected]

standard deviation of observed data (dimension: nt x nytrue, row-major)

Definition at line 389 of file [edata.h](#).

10.6.4.10 observedEvents

std::vector<[realtype](#)> observedEvents [protected]

observed events (dimension: nmaxevents x nztrue, row-major)

Definition at line 392 of file [edata.h](#).

10.6.4.11 observedEventsStdDev

std::vector<[realtype](#)> observedEventsStdDev [protected]

standard deviation of observed events/roots (dimension: nmaxevents x nztrue, row-major)

Definition at line 395 of file [edata.h](#).

10.7 ForwardProblem Class Reference

The [ForwardProblem](#) class groups all functions for solving the backwards problem. Has only static members.

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

Public Member Functions

- `ForwardProblem (ReturnData *rdata, const ExpData *edata, Model *model, Solver *solver)`
- `void workForwardProblem ()`
- `realtype getTime () const`
- `AmiVectorArray const & getStateSensitivity () const`
- `AmiVectorArray const & getStatesAtDiscontinuities () const`
- `AmiVectorArray const & getRHSAtDiscontinuities () const`
- `AmiVectorArray const & getRHSBeforeDiscontinuities () const`
- `std::vector< int > const & getNumberOfRoots () const`
- `std::vector< realtype > const & getDiscontinuities () const`
- `std::vector< int > const & getRootIndexes () const`
- `std::vector< realtype > const & getDjydx () const`
- `std::vector< realtype > const & getDzwdx () const`
- `int getRootCounter () const`
- `AmiVector * getStatePointer ()`
- `AmiVector * getStateDerivativePointer ()`
- `AmiVectorArray * getStateSensitivityPointer ()`
- `AmiVectorArray * getStateDerivativeSensitivityPointer ()`

Public Attributes

- `Model * model`
- `ReturnData * rdata`
- `Solver * solver`
- `const ExpData * edata`

10.7.1 Detailed Description

Definition at line 23 of file forwardproblem.h.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 ForwardProblem()

```
ForwardProblem (
    ReturnData * rdata,
    const ExpData * edata,
    Model * model,
    Solver * solver )
```

Constructor

Parameters

<code>rdata</code>	pointer to <code>ReturnData</code> instance
<code>edata</code>	pointer to <code>ExpData</code> instance
<code>model</code>	pointer to <code>Model</code> instance
<code>solver</code>	pointer to <code>Solver</code> instance

Definition at line 42 of file forwardproblem.cpp.

10.7.3 Member Function Documentation

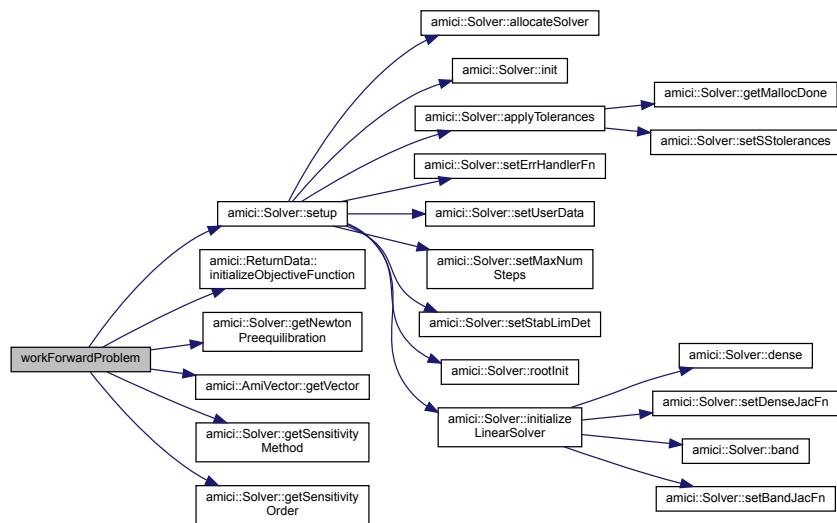
10.7.3.1 workForwardProblem()

```
void workForwardProblem ( )
```

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

Definition at line 77 of file forwardproblem.cpp.

Here is the call graph for this function:



10.7.3.2 getTime()

```
realtypetime ( ) const
```

accessor for t

Returns`t`

Definition at line 37 of file forwardproblem.h.

Here is the caller graph for this function:



10.7.3.3 getStateSensitivity()

```
AmiVectorArray const& getStateSensitivity() const
```

accessor for sx

Returns`sx`

Definition at line 44 of file forwardproblem.h.

10.7.3.4 getStatesAtDiscontinuities()

```
AmiVectorArray const& getStatesAtDiscontinuities() const
```

accessor for x_disc

Returns`x_disc`

Definition at line 51 of file forwardproblem.h.

10.7.3.5 getRHSAtDiscontinuities()

```
AmiVectorArray const& getRHSAtDiscontinuities ( ) const  
accessor for xdot_disc
```

Returns

xdot_disc

Definition at line 58 of file forwardproblem.h.

10.7.3.6 getRHSBeforeDiscontinuities()

```
AmiVectorArray const& getRHSBeforeDiscontinuities ( ) const  
accessor for xdot_old_disc
```

Returns

xdot_old_disc

Definition at line 65 of file forwardproblem.h.

10.7.3.7 getNumberOfRoots()

```
std::vector<int> const& getNumberOfRoots ( ) const  
accessor for nroots
```

Returns

nroots

Definition at line 72 of file forwardproblem.h.

10.7.3.8 getDiscontinuities()

```
std::vector<realtyp> const& getDiscontinuities ( ) const  
accessor for discs
```

Returns

discs

Definition at line 79 of file forwardproblem.h.

10.7.3.9 getRootIndexes()

```
std::vector<int> const& getRootIndexes () const
```

accessor for rootidx

Returns

rootidx

Definition at line 86 of file forwardproblem.h.

10.7.3.10 getDJydx()

```
std::vector<realtyp> const& getDJydx () const
```

accessor for dJydx

Returns

dJydx

Definition at line 93 of file forwardproblem.h.

10.7.3.11 getDJzdx()

```
std::vector<realtyp> const& getDJzdx () const
```

accessor for dJzdx

Returns

dJzdx

Definition at line 100 of file forwardproblem.h.

10.7.3.12 getRootCounter()

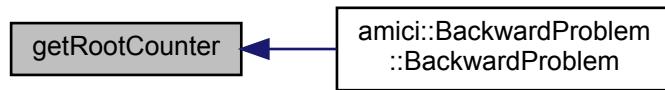
```
int getRootCounter ( ) const  
accessor for iroot
```

Returns

iroot

Definition at line 107 of file forwardproblem.h.

Here is the caller graph for this function:



10.7.3.13 getStatePointer()

```
AmiVector* getStatePointer ( )
```

accessor for pointer to x

Returns

&x

Definition at line 114 of file forwardproblem.h.

10.7.3.14 getStateDerivativePointer()

```
AmiVector* getStateDerivativePointer ( )
```

accessor for pointer to dx

Returns

&dx

Definition at line 121 of file forwardproblem.h.

10.7.3.15 getStateSensitivityPointer()

```
AmiVectorArray* getStateSensitivityPointer ( )
```

accessor for pointer to sx

Returns

&sx

Definition at line 128 of file forwardproblem.h.

10.7.3.16 getStateDerivativeSensitivityPointer()

```
AmiVectorArray* getStateDerivativeSensitivityPointer ( )
```

accessor for pointer to sdx

Returns

&sdx

Definition at line 135 of file forwardproblem.h.

10.7.4 Member Data Documentation**10.7.4.1 model**

```
Model* model
```

pointer to model instance

Definition at line 140 of file forwardproblem.h.

10.7.4.2 rdata

```
ReturnData* rdata
```

pointer to return data instance

Definition at line 142 of file forwardproblem.h.

10.7.4.3 solver

`Solver*` solver

pointer to solver instance

Definition at line 144 of file forwardproblem.h.

10.7.4.4 edata

`const ExpData*` edata

pointer to experimental data instance

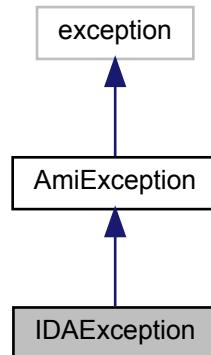
Definition at line 146 of file forwardproblem.h.

10.8 IDAException Class Reference

ida exception handler class

`#include <exception.h>`

Inheritance diagram for IDAException:



Public Member Functions

- `IDAException (const int error_code, const char *function)`

10.8.1 Detailed Description

Definition at line 129 of file exception.h.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 IDAException()

```
IDAEException (
    const int error_code,
    const char * function )
```

constructor

Parameters

<i>error_code</i>	error code returned by ida function
<i>function</i>	ida function name

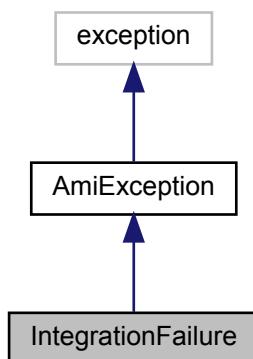
Definition at line 135 of file exception.h.

10.9 IntegrationFailure Class Reference

integration failure exception for the forward problem this exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user

```
#include <exception.h>
```

Inheritance diagram for IntegrationFailure:



Public Member Functions

- [IntegrationFailure \(int code, realtype t\)](#)

Public Attributes

- int `error_code`
- `realtypes` `time`

10.9.1 Detailed Description

Definition at line 144 of file exception.h.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 `IntegrationFailure()`

```
IntegrationFailure (
    int code,
    realtype t )
```

constructor

Parameters

<code>code</code>	error code returned by cvode/ida
<code>t</code>	time of integration failure

Definition at line 154 of file exception.h.

10.9.3 Member Data Documentation

10.9.3.1 `error_code`

```
int error_code
```

error code returned by cvode/ida

Definition at line 147 of file exception.h.

10.9.3.2 `time`

```
realtypes time
```

time of integration failure

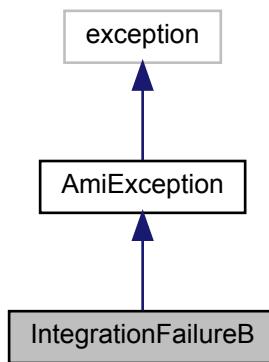
Definition at line 149 of file exception.h.

10.10 IntegrationFailureB Class Reference

integration failure exception for the backward problem this exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user

```
#include <exception.h>
```

Inheritance diagram for IntegrationFailureB:



Public Member Functions

- [IntegrationFailureB \(int code, realtype t\)](#)

Public Attributes

- int [error_code](#)
- [realtype time](#)

10.10.1 Detailed Description

Definition at line 166 of file exception.h.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 IntegrationFailureB()

```
IntegrationFailureB (
    int code,
    realtype t )
```

constructor

Parameters

<i>code</i>	error code returned by cvode/ida
<i>t</i>	time of integration failure

Definition at line 176 of file exception.h.

10.10.3 Member Data Documentation

10.10.3.1 `error_code`

```
int error_code
```

error code returned by cvode/ida

Definition at line 169 of file exception.h.

10.10.3.2 `time`

```
realtype time
```

time of integration failure

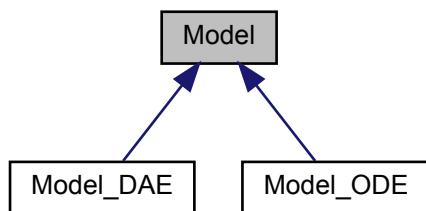
Definition at line 171 of file exception.h.

10.11 Model Class Reference

The [Model](#) class represents an AMICI ODE model. The model can compute various model related quantities based on symbolically generated code.

```
#include <model.h>
```

Inheritance diagram for Model:



Public Member Functions

- `Model ()`
- `Model (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 nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, amici::SecondOrderMode o2mode, const std::vector< amici::realtype > &p, std::vector< amici::realtype > k, const std::vector< int > &plist, std::vector< amici::realtype > idlist, std::vector< int > z2event)`
- `Model (Model const &other)`
- `virtual ~Model ()`
- `Model & operator= (Model const &other)=delete`
- `virtual Model * clone () const =0`

Clone this instance.

 - `virtual std::unique_ptr< Solver > getSolver ()=0`
 - `virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root)=0`
 - `virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot)=0`
 - `virtual void fsxdot (realtype t, AmiVector *x, AmiVector *dx, int ip, AmiVector *sx, AmiVector *sdx, AmiVector *sxdot)=0`
 - `virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J)=0`
 - `virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J)=0`
 - `virtual void fJDiag (realtype t, AmiVector *Jdiag, realtype cj, AmiVector *x, AmiVector *dx)=0`
 - `virtual void fdxdotdp (realtype t, AmiVector *x, AmiVector *dx)=0`
 - `virtual void fJv (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj)=0`
 - `void fx0 (AmiVector *x)`
 - `void fx0_fixedParameters (AmiVector *x)`
 - `virtual void fdx0 (AmiVector *x0, AmiVector *dx0)`
 - `void fsx0 (AmiVectorArray *sx, const AmiVector *x)`
 - `void fsx0_fixedParameters (AmiVectorArray *sx, const AmiVector *x)`
 - `virtual void fsdx0 ()`
 - `void fstau (const realtype t, const int ie, const AmiVector *x, const AmiVectorArray *sx)`
 - `void fy (int it, ReturnData *rdata)`
 - `void fdydp (const int it, ReturnData *rdata)`
 - `void fdydx (const int it, ReturnData *rdata)`
 - `void fz (const int nroots, const int ie, const realtype t, const AmiVector *x, ReturnData *rdata)`
 - `void fsz (const int nroots, const int ie, const realtype t, const AmiVector *x, const AmiVectorArray *sx, ReturnData *rdata)`
 - `void frz (const int nroots, const int ie, const realtype t, const AmiVector *x, ReturnData *rdata)`
 - `void fsrz (const int nroots, const int ie, const realtype t, const AmiVector *x, const AmiVectorArray *sx, ReturnData *rdata)`
 - `void fdzdp (const realtype t, const int ie, const AmiVector *x)`
 - `void fdzdx (const realtype t, const int ie, const AmiVector *x)`
 - `void fdrzdp (const realtype t, const int ie, const AmiVector *x)`
 - `void fdrzdx (const realtype t, const int ie, const AmiVector *x)`
 - `void fdeletax (const int ie, const realtype t, const AmiVector *x, const AmiVector *xdot, const AmiVector *xdot←_old)`
 - `void fdeletasx (const int ie, const realtype t, const AmiVector *x, const AmiVectorArray *sx, const AmiVector *xdot, const AmiVector *xdot_old)`
 - `void fdeletaxB (const int ie, const realtype t, const AmiVector *x, const AmiVector *xB, const AmiVector *xdot, const AmiVector *xdot_old)`
 - `void fdeletaqB (const int ie, const realtype t, const AmiVector *x, const AmiVector *xB, const AmiVector *xdot, const AmiVector *xdot_old)`
 - `void fsigmay (const int it, ReturnData *rdata, const ExpData *edata)`
 - `void fdsigmaydp (const int it, ReturnData *rdata, const ExpData *edata)`
 - `void fsigmaz (const realtype t, const int ie, const int *nroots, ReturnData *rdata, const ExpData *edata)`
 - `void fdsigmazdp (const realtype t, const int ie, const int *nroots, ReturnData *rdata, const ExpData *edata)`

- void **fJy** (const int it, **ReturnData** *rdata, const **ExpData** *edata)
- void **fJz** (const int nroots, **ReturnData** *rdata, const **ExpData** *edata)
- void **fJrz** (const int nroots, **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJydy** (const int it, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJydsigma** (const int it, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJzdz** (const int nroots, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJzdsigma** (const int nroots, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJrzdz** (const int nroots, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fdJrzdsigma** (const int nroots, const **ReturnData** *rdata, const **ExpData** *edata)
- void **fsy** (const int it, **ReturnData** *rdata)
- void **fsz_tf** (const int *nroots, const int ie, **ReturnData** *rdata)
- void **fsJy** (const int it, const std::vector< **realtypes** > &dJydx, **ReturnData** *rdata)
- void **fdJydp** (const int it, const **ExpData** *edata, **ReturnData** *rdata)
- void **fdJydx** (std::vector< **realtypes** > *dJydx, const int it, const **ExpData** *edata, const **ReturnData** *rdata)
- void **fsJz** (const int nroots, const std::vector< **realtypes** > &dJzdx, **AmiVectorArray** *sx, **ReturnData** *rdata)
- void **fdJzdp** (const int nroots, **realtypes** t, const **ExpData** *edata, const **ReturnData** *rdata)
- void **fdJzdx** (std::vector< **realtypes** > *dJzdx, const int nroots, **realtypes** t, const **ExpData** *edata, const **ReturnData** *rdata)
- void **initialize** (**AmiVector** *x, **AmiVector** *dx)
- void **initializeStates** (**AmiVector** *x)
- void **initHeaviside** (**AmiVector** *x, **AmiVector** *dx)
- int **nplist** () const

number of parameters wrt to which sensitivities are computed
- int **np** () const

total number of model parameters
- int **nk** () const

number of constants
- const double * **k** () const

fixed parameters
- int **nMaxEvent** () const

Get nmaxevent.
- void **setNMaxEvent** (int nmaxevent)

Set nmaxevent.
- int **nt** () const

Get number of timepoints.
- std::vector< **ParameterScaling** > const & **getParameterScale** () const

Get ParameterScale for each parameter.
- void **setParameterScale** (**ParameterScaling** pscale)

Set ParameterScale for each parameter.
- void **setParameterScale** (const std::vector< **ParameterScaling** > &pscale)

Set ParameterScale for each parameter.
- std::vector< **realtypes** > const & **getParameters** () const

Get the parameter vector.
- void **setParameters** (std::vector< **realtypes** > const &p)

Sets the parameter vector.
- std::vector< **realtypes** > const & **getUnscaledParameters** () const

Gets parameters with transformation according to ParameterScale applied.
- std::vector< **realtypes** > const & **getFixedParameters** () const

Gets the fixedParameter member.
- void **setFixedParameters** (std::vector< **realtypes** > const &k)

Sets the fixedParameter member.
- **realtypes getFixedParameterById** (std::string const &par_id) const

- **realtype getFixedParameterByName** (std::string const &par_name) const
Get value of fixed parameter with the specified Id.
- void **setFixedParameterById** (std::string const &par_id, realtype value)
Set value of first fixed parameter with the specified id.
- int **setFixedParametersByIdRegex** (std::string const &par_id_regex, realtype value)
Set values of all fixed parameters with the id matching the specified regex.
- void **setFixedParameterByName** (std::string const &par_name, realtype value)
Set value of first fixed parameter with the specified name.,
- int **setFixedParametersByNameRegex** (std::string const &par_name_regex, realtype value)
Set value of all fixed parameters with name matching the specified regex.,
- std::vector< realtype > const & **getTimepoints** () const
Get the timepoint vector.
- void **setTimepoints** (std::vector< realtype > const &ts)
Set the timepoint vector.
- std::vector< bool > const & **getStateIsNonNegative** () const
gets flags indicating whether states should be treated as non-negative
- void **setStateIsNonNegative** (std::vector< bool > const &stateIsNonNegative)
sets flags indicating whether states should be treated as non-negative
- double **t** (int idx) const
Get timepoint for given index.
- std::vector< int > const & **getParameterList** () const
Get the list of parameters for which sensitivities are computed.
- void **setParameterList** (std::vector< int > const &plist)
Set the list of parameters for which sensitivities are computed.
- std::vector< realtype > const & **getInitialStates** () const
Get the initial states.
- void **setInitialStates** (std::vector< realtype > const &x0)
Set the initial states.
- std::vector< realtype > const & **getInitialStateSensitivities** () const
Get the initial states sensitivities.
- void **setInitialStateSensitivities** (std::vector< realtype > const &sx0)
Set the initial state sensitivities.
- double **t0** () const
get simulation start time
- void **setT0** (double t0)
set simulation start time
- int **plist** (int pos) const
entry in parameter list
- void **requireSensitivitiesForAllParameters** ()
Require computation of sensitivities for all parameters p [0..np[in natural order.
- void **fw** (const realtype t, const realtype *x)
Recurring terms in xdot.
- void **fdwdp** (const realtype t, const realtype *x)
Recurring terms in xdot, parameter derivative.
- void **fdwdx** (const realtype t, const realtype *x)
Recurring terms in xdot, state derivative.
- void **fres** (const int it, ReturnData *rdata, const ExpData *edata)
- void **fchi2** (const int it, ReturnData *rdata)
- void **fsres** (const int it, ReturnData *rdata, const ExpData *edata)

- void **fFIM** (const int it, **ReturnData** *rdata)
- void **updateHeaviside** (const std::vector< int > &rootsfound)
- void **updateHeavisideB** (const int *rootsfound)
- **realtype** **gett** (const int it, const **ReturnData** *rdata) const
- int **checkFinite** (const int N, const **realtype** *array, const char *fun) const

Check if the given array has only finite elements. If not try to give hints by which other fields this could be caused.
- virtual bool **hasParameterNames** () const

Reports whether the model has parameter names set.
- virtual std::vector< std::string > **getParameterNames** () const

Get names of the model parameters.
- virtual bool **hasStateNames** () const

Reports whether the model has state names set.
- virtual std::vector< std::string > **getStateNames** () const

Get names of the model states.
- virtual bool **hasFixedParameterNames** () const

Reports whether the model has fixed parameter names set.
- virtual std::vector< std::string > **getFixedParameterNames** () const

Get names of the fixed model parameters.
- virtual bool **hasObservableNames** () const

Reports whether the model has observable names set.
- virtual std::vector< std::string > **getObservableNames** () const

Get names of the observables.
- virtual bool **hasParameterIds** () const

Reports whether the model has parameter ids set.
- virtual std::vector< std::string > **getParameterIds** () const

Get ids of the model parameters.
- **realtype** **getParameterById** (std::string const &par_id) const

Get value of first model parameter with the specified id.
- **realtype** **getParameterByName** (std::string const &par_name) const

Get value of first model parameter with the specified name.,
- void **setParameterById** (std::string const &par_id, **realtype** value)

Set value of first model parameter with the specified id.
- int **setParametersByIdRegex** (std::string const &par_id_regex, **realtype** value)

Set all values of model parameters with ids matching the specified regex.
- void **setParameterByName** (std::string const &par_name, **realtype** value)

Set value of first model parameter with the specified name.
- int **setParametersByNameRegex** (std::string const &par_name_regex, **realtype** value)

Set all values of all model parameters with names matching the specified regex.
- virtual bool **hasStatelids** () const

Reports whether the model has state ids set.
- virtual std::vector< std::string > **getStatelids** () const

Get ids of the model states.
- virtual bool **hasFixedParameterIds** () const

Reports whether the model has fixed parameter ids set.
- virtual std::vector< std::string > **getFixedParameterIds** () const

Get ids of the fixed model parameters.
- virtual bool **hasObservableIds** () const

Reports whether the model has observable ids set.
- virtual std::vector< std::string > **getObservableIds** () const

Get ids of the observables.
- void **setSteadyStateSensitivityMode** (const **SteadyStateSensitivityMode** mode)

- sets the mode how sensitivities are computed in the steadystate simulation*
- `SteadyStateSensitivityMode getSteadyStateSensitivityMode () const`
gets the mode how sensitivities are computed in the steadystate simulation
- `void setReinitializeFixedParameterInitialStates (bool flag)`
set whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation
- `bool getReinitializeFixedParameterInitialStates () const`
get whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation

Public Attributes

- `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 SecondOrderMode o2mode`
- `const std::vector< int > z2event`
- `const std::vector< realtype > idlist`
- `std::vector< realtype > sigmay`
- `std::vector< realtype > dsigmaydp`
- `std::vector< realtype > sigmaz`
- `std::vector< realtype > dsigmazdp`
- `std::vector< realtype > dJydp`
- `std::vector< realtype > dJzdp`
- `std::vector< realtype > deltax`
- `std::vector< realtype > deltasx`
- `std::vector< realtype > deltaxB`
- `std::vector< realtype > deltaqB`
- `std::vector< realtype > dxdotdp`

Protected Member Functions

- `void initializeVectors ()`
Set the nplist-dependent vectors to their proper sizes.
- `virtual void fx0 (realtype *x0, const realtype t, const realtype *p, const realtype *k)`
- `virtual void fx0_fixedParameters (realtype *x0, const realtype t, const realtype *p, const realtype *k)`
- `virtual void fsx0_fixedParameters (realtype *sx0, const realtype t, const realtype *x0, const realtype *p, const realtype *k, const int ip)`
- `virtual void fsx0 (realtype *sx0, const realtype t, const realtype *x0, const realtype *p, const realtype *k, const int ip)`
- `virtual void fstau (realtype *stau, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip, const int ie)`

- virtual void **fy** (realtype *y, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)
- virtual void **fddydp** (realtype *dydp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *w, const realtype *dwdp)
- virtual void **fdydx** (realtype *dydx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)
- virtual void **fz** (realtype *z, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void **fsz** (realtype *sz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void **frz** (realtype *rz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void **fsrz** (realtype *srz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void **fdzdp** (realtype *dzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void **fdzdx** (realtype *dzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void **fdrzdp** (realtype *drzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void **fdrzdx** (realtype *drzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void **fdeletax** (realtype *deltax, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old)
- virtual void **fdeletaxs** (realtype *deltasx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *sx, const realtype *stau)
- virtual void **fdeletaxB** (realtype *deltaxB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void **fdeletaqB** (realtype *deltaqB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void **fsigmay** (realtype *sigmay, const realtype t, const realtype *p, const realtype *k)
- virtual void **fdsigmaydp** (realtype *dsigmaydp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void **fsigmaz** (realtype *sigmaz, const realtype t, const realtype *p, const realtype *k)
- virtual void **fdsigmazdp** (realtype *dsigmazdp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void **fJy** (realtype *nllh, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void **fJz** (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void **fJrz** (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz)
- virtual void **fdJydy** (realtype *dJydy, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void **fdJydsigma** (realtype *dJydsigma, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void **fdJzdz** (realtype *dJzdz, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void **fdJzdsigma** (realtype *dJzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void **fdJrzdz** (realtype *dJrzdz, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)
- virtual void **fdJrzdsigma** (realtype *dJrzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)

- virtual void `fw` (`realtype *w`, const `realtype t`, const `realtype *x`, const `realtype *p`, const `realtype *k`, const `realtype *h`)
- virtual void `fdwdp` (`realtype *dwdp`, const `realtype t`, const `realtype *x`, const `realtype *p`, const `realtype *k`, const `realtype *h`, const `realtype *w`)
- virtual void `fdwdx` (`realtype *dwdx`, const `realtype t`, const `realtype *x`, const `realtype *p`, const `realtype *k`, const `realtype *h`, const `realtype *w`)
- void `getmy` (const int `it`, const `ExpData *edata`)
- void `getmz` (const int `nroots`, const `ExpData *edata`)
- const `realtype * gety` (const int `it`, const `ReturnData *rdata`) const
- const `realtype * getx` (const int `it`, const `ReturnData *rdata`) const
- const `realtype * getsx` (const int `it`, const `ReturnData *rdata`) const
- const `realtype * getz` (const int `nroots`, const `ReturnData *rdata`) const
- const `realtype * getrz` (const int `nroots`, const `ReturnData *rdata`) const
- const `realtype * getsz` (const int `nroots`, const int `ip`, const `ReturnData *rdata`) const
- const `realtype * getsrz` (const int `nroots`, const int `ip`, const `ReturnData *rdata`) const
- virtual bool `isFixedParameterStateReinitializationAllowed` () const
- `N_Vector computeX_pos` (`N_Vector x`)

computes nonnegative state vector according to `stateIsNonNegative` if `anyStateNonNegative` is set to false, i.e., all entries in `stateIsNonNegative` are false, this function directly returns `x`, otherwise all entries of `x` are copied in to `x_pos_tmp` and negative values are replaced by 0 where applicable

Protected Attributes

- `SlsMat J = nullptr`
- `std::vector< realtype > my`
- `std::vector< realtype > mz`
- `std::vector< realtype > dJydy`
- `std::vector< realtype > dJydsigma`
- `std::vector< realtype > dJzdz`
- `std::vector< realtype > dJzdsigma`
- `std::vector< realtype > dJrzdz`
- `std::vector< realtype > dJrzdsigma`
- `std::vector< realtype > dzdx`
- `std::vector< realtype > dzdp`
- `std::vector< realtype > drzdx`
- `std::vector< realtype > drzdp`
- `std::vector< realtype > dydp`
- `std::vector< realtype > dydx`
- `std::vector< realtype > w`
- `std::vector< realtype > dwdx`
- `std::vector< realtype > dwdp`
- `std::vector< realtype > M`
- `std::vector< realtype > stau`
- `std::vector< realtype > h`
- `std::vector< realtype > unscaledParameters`
- `std::vector< realtype > originalParameters`
- `std::vector< realtype > fixedParameters`
- `std::vector< int > plist_`
- `std::vector< double > x0data`
- `std::vector< realtype > sx0data`
- `std::vector< realtype > ts`
- `std::vector< bool > stateIsNonNegative`
- `bool anyStateNonNegative = false`
- `AmiVector x_pos_tmp`

- int `nmaxevent` = 10
- std::vector< ParameterScaling > `psscale`
- double `tstart` = 0.0
- SteadyStateSensitivityMode `steadyStateSensitivityMode` = SteadyStateSensitivityMode::newtonOnly
- bool `reinitializeFixedParameterInitialStates` = false

Friends

- template<class Archive >
void `boost::serialization::serialize` (Archive &r, Model &r, const unsigned int version)
Serialize Model (see boost::serialization::serialize)
- bool `operator==` (const Model &a, const Model &b)
Check equality of data members.

10.11.1 Detailed Description

Definition at line 40 of file model.h.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 Model() [1/3]

`Model ()`

default constructor

Definition at line 43 of file model.h.

10.11.2.2 Model() [2/3]

```
Model (
    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 nw,
    const int ndwdx,
    const int ndwdp,
    const int nnz,
    const int ubw,
    const int lbw,
    amici::SecondOrderMode o2mode,
    const std::vector< amici::realtype > & p,
    std::vector< amici::realtype > k,
    const std::vector< int > & plist,
    std::vector< amici::realtype > idlist,
    std::vector< int > z2event )
```

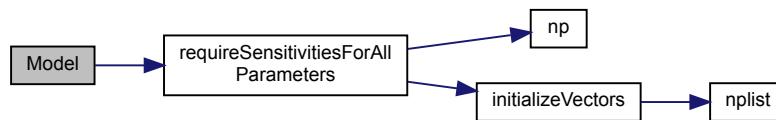
constructor with model dimensions

Parameters

<i>nx</i>	number of state variables
<i>nxtrue</i>	number of state variables of the non-augmented model
<i>ny</i>	number of observables
<i>nytrue</i>	number of observables of the non-augmented model
<i>nz</i>	number of event observables
<i>nztrue</i>	number of event observables of the non-augmented model
<i>ne</i>	number of events
<i>nJ</i>	number of objective functions
<i>nw</i>	number of repeating elements
<i>ndwdx</i>	number of nonzero elements in the x derivative of the repeating elements
<i>ndwdp</i>	number of nonzero elements in the p derivative of the repeating elements
<i>nnz</i>	number of nonzero elements in Jacobian
<i>ubw</i>	upper matrix bandwidth in the Jacobian
<i>lbw</i>	lower matrix bandwidth in the Jacobian
<i>o2mode</i>	second order sensitivity mode
<i>p</i>	parameters
<i>k</i>	constants
<i>plist</i>	indexes wrt to which sensitivities are to be computed
<i>idlist</i>	indexes indicating algebraic components (DAE only)
<i>z2event</i>	mapping of event outputs to events

Definition at line 626 of file model.cpp.

Here is the call graph for this function:

**10.11.2.3 Model() [3/3]**

```
Model (
    Model const & other )
```

Copy constructor

Parameters

<i>other</i>	object to copy from
--------------	---------------------

Returns

Definition at line 703 of file model.cpp.

10.11.2.4 ~Model()

```
~Model ( ) [virtual]
```

destructor

Definition at line 764 of file model.cpp.

10.11.3 Member Function Documentation**10.11.3.1 operator=()**

```
Model& operator= (
    Model const & other ) [delete]
```

Copy assignment is disabled until const members are removed

Parameters

<i>other</i>	object to copy from
--------------	---------------------

Returns**10.11.3.2 clone()**

```
virtual Model* clone ( ) const [pure virtual]
```

Returns

The clone

10.11.3.3 getSolver()

```
virtual std::unique_ptr<Solver> getSolver () [pure virtual]
```

Retrieves the solver object

Returns

The [Solver](#) instance

Implemented in [Model_DAE](#), and [Model_ODE](#).

10.11.3.4 froot()

```
virtual void froot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    realtype * root ) [pure virtual]
```

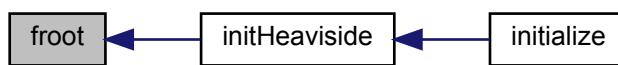
Root function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>root</i>	array to which values of the root function will be written

Implemented in [Model_ODE](#), and [Model_DAE](#).

Here is the caller graph for this function:



10.11.3.5 fxdot()

```
virtual void fxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot ) [pure virtual]
```

Residual function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	array to which values of the residual function will be written

Implemented in [Model_ODE](#), and [Model_DAE](#).

10.11.3.6 fsxdot()

```
virtual void fsxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    int ip,
    AmiVector * sx,
    AmiVector * sdx,
    AmiVector * sxdot ) [pure virtual]
```

Sensitivity Residual function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>ip</i>	parameter index
<i>sx</i>	sensitivity state
<i>sdx</i>	time derivative of sensitivity state (DAE only)
<i>sxdot</i>	array to which values of the sensitivity residual function will be written

Implemented in [Model_ODE](#), and [Model_DAE](#).

10.11.3.7 fJ()

```
virtual void fJ (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    DlsMat J ) [pure virtual]
```

Dense Jacobian function

Parameters

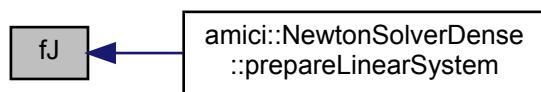
<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)

Parameters

<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	dense matrix to which values of the jacobian will be written

Implemented in [Model_DAE](#), and [Model_ODE](#).

Here is the caller graph for this function:

**10.11.3.8 fJSparse()**

```

virtual void fJSparse (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    SlsMat J ) [pure virtual]

```

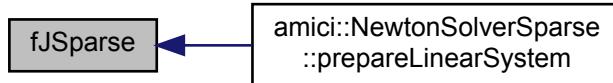
Sparse Jacobian function

Parameters

<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	sparse matrix to which values of the Jacobian will be written

Implemented in [Model_ODE](#), and [Model_DAE](#).

Here is the caller graph for this function:



10.11.3.9 fJDiag()

```

virtual void fJDiag (
    realtype t,
    AmiVector * Jdiag,
    realtype cj,
    AmiVector * x,
    AmiVector * dx ) [pure virtual]
  
```

Diagonal Jacobian function

Parameters

<i>t</i>	time
<i>Jdiag</i>	array to which the diagonal of the Jacobian will be written
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)

Returns

flag indicating successful evaluation

Implemented in [Model_ODE](#), and [Model_DAE](#).

Here is the caller graph for this function:



10.11.3.10 fxdotdp()

```
virtual void fxdotdp (
    realtype t,
    AmiVector * x,
    AmiVector * dx ) [pure virtual]
```

parameter derivative of residual function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)

Returns

flag indicating successful evaluation

Implemented in [Model_ODE](#), and [Model_DAE](#).

Here is the caller graph for this function:



10.11.3.11 fJv()

```
virtual void fJv (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    AmiVector * v,
    AmiVector * nJv,
    realtype cj ) [pure virtual]
```

Jacobian multiply function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>nJv</i>	multiplication vector (unused)
<i>cj</i>	array to which result of multiplication will be written
<i>cj</i>	scaling factor (inverse of timestep, DAE only)

Implemented in [Model_ODE](#), and [Model_DAE](#).

Here is the caller graph for this function:



10.11.3.12 fx0() [1/2]

```
void fx0 (
    AmiVector * x )
```

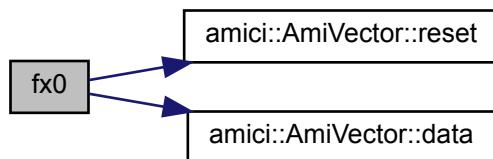
Initial states

Parameters

x	pointer to state variables
---	----------------------------

Definition at line 785 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.13 fxFixedParameters() [1/2]

```
void fxFixedParameters (
    AmiVector * x )
```

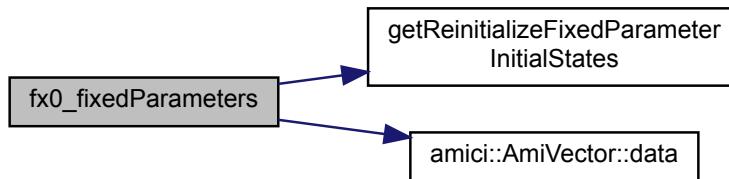
Sets only those initial states that are specified via fixedParmeters

Parameters

x	pointer to state variables
---	----------------------------

Definition at line 790 of file model.cpp.

Here is the call graph for this function:



10.11.3.14 fdx0()

```
void fdx0 (
    AmiVector * x0,
    AmiVector * dx0 ) [virtual]
```

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

Parameters

x0	Vector with the initial states
dx0	Vector to which the initial derivative states will be written (only DAE)

Definition at line 803 of file model.cpp.

Here is the caller graph for this function:



10.11.3.15 fsx0() [1/2]

```
void fsx0 (
    AmiVectorArray * sx,
    const AmiVector * x )
```

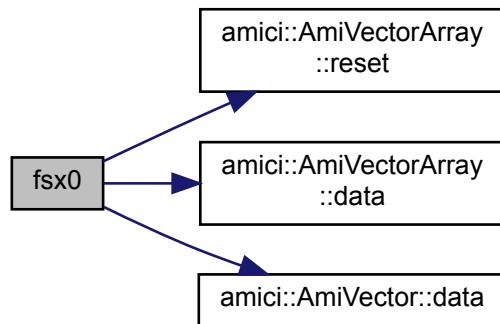
Initial value for initial state sensitivities

Parameters

<code>sx</code>	pointer to state sensitivity variables
<code>x</code>	pointer to state variables

Definition at line 805 of file model.cpp.

Here is the call graph for this function:



10.11.3.16 fsx0_fixedParameters() [1/2]

```
void fsx0_fixedParameters (
    AmiVectorArray * sx,
    const AmiVector * x )
```

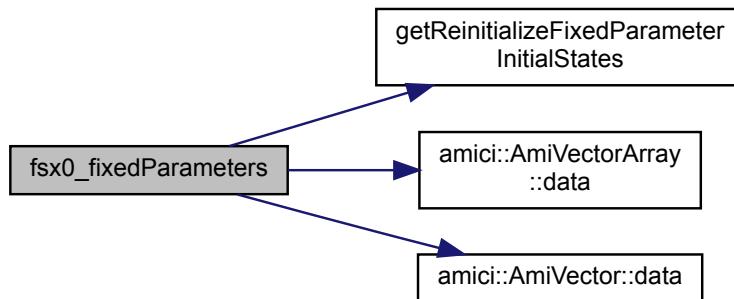
Sets only those initial states sensitivities that are affected from fx0 fixedParmeters

Parameters

sx	pointer to state sensitivity variables
x	pointer to state variables

Definition at line 795 of file model.cpp.

Here is the call graph for this function:



10.11.3.17 fsdx0()

```
void fsdx0 ( ) [virtual]
```

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

Definition at line 811 of file model.cpp.

10.11.3.18 fstau() [1/2]

```
void fstau (
    const realtype t,
    const int ie,
    const AmiVector * x,
    const AmiVectorArray * sx )
```

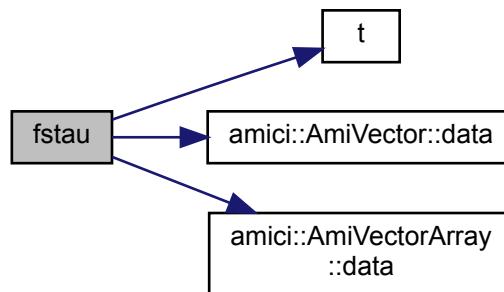
Sensitivity of event timepoint, total derivative

Parameters

<i>t</i>	current timepoint
<i>ie</i>	event index
<i>x</i>	pointer to state variables
<i>sx</i>	pointer to state sensitivity variables

Definition at line 813 of file model.cpp.

Here is the call graph for this function:

**10.11.3.19 fy() [1/2]**

```
void fy (
    int it,
    ReturnData * rdata )
```

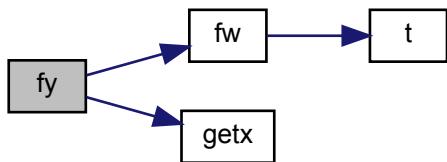
Observables / measurements

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Definition at line 820 of file model.cpp.

Here is the call graph for this function:



10.11.3.20 fdydp() [1/2]

```
void fdydp (
    const int it,
    ReturnData * rdata )
```

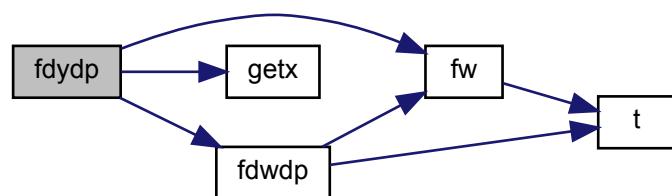
partial derivative of observables y w.r.t. model parameters p

Parameters

<code>it</code>	timepoint index
<code>rdata</code>	pointer to return data instance

Definition at line 827 of file model.cpp.

Here is the call graph for this function:



10.11.3.21 fdydx() [1/2]

```
void fdydx (
    const int it,
    ReturnData * rdata )
```

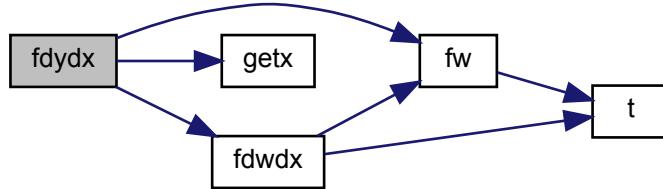
partial derivative of observables y w.r.t. state variables x

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Definition at line 848 of file model.cpp.

Here is the call graph for this function:

**10.11.3.22 fz() [1/2]**

```
void fz (
    const int nroots,
    const int ie,
    const realtype t,
    const AmiVector * x,
    ReturnData * rdata )
```

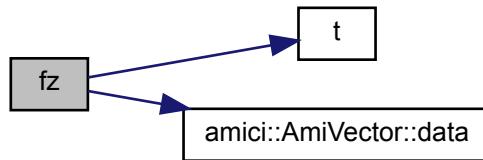
Event-resolved output

Parameters

<i>nroots</i>	number of events for event index
<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>rdata</i>	pointer to return data instance

Definition at line 858 of file model.cpp.

Here is the call graph for this function:



10.11.3.23 fsz() [1/2]

```
void fsz (
    const int nroots,
    const int ie,
    const realtype t,
    const AmiVector * x,
    const AmiVectorArray * sx,
    ReturnData * rdata )
```

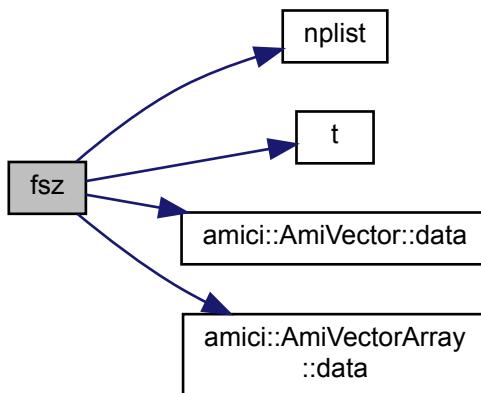
Sensitivity of z, total derivative

Parameters

<i>nroots</i>	number of events for event index
<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>sx</i>	current state sensitivities
<i>rdata</i>	pointer to return data instance

Definition at line 862 of file model.cpp.

Here is the call graph for this function:



10.11.3.24 frz() [1/2]

```
void frz (
    const int nroots,
    const int ie,
    const realtype t,
    const AmiVector * x,
    ReturnData * rdata )
```

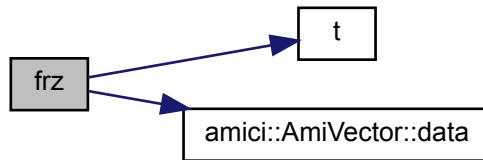
Event root function of events (equal to froot but does not include non-output events)

Parameters

<i>nroots</i>	number of events for event index
<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>rdata</i>	pointer to return data instance

Definition at line 868 of file model.cpp.

Here is the call graph for this function:



10.11.3.25 fsrz() [1/2]

```
void fsrz (
    const int nroots,
    const int ie,
    const realltype t,
    const AmiVector * x,
    const AmiVectorArray * sx,
    ReturnData * rdata )
```

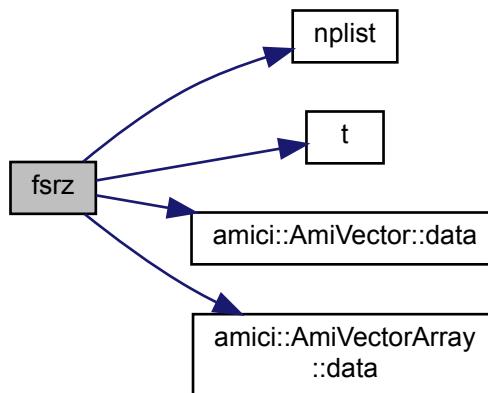
Sensitivity of rz, total derivative

Parameters

<i>nroots</i>	number of events for event index
<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>sx</i>	current state sensitivities
<i>rdata</i>	pointer to return data instance

Definition at line 872 of file model.cpp.

Here is the call graph for this function:



10.11.3.26 fdzdp() [1/2]

```
void fdzdp (
    const realtype t,
    const int ie,
    const AmiVector * x )
```

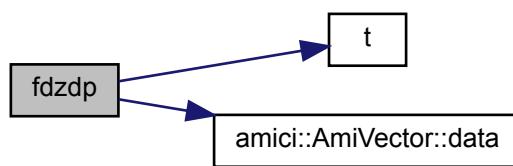
partial derivative of event-resolved output z w.r.t. to model parameters p

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state

Definition at line 878 of file model.cpp.

Here is the call graph for this function:



10.11.3.27 fdzdx() [1/2]

```
void fdzdx (
    const realltype t,
    const int ie,
    const AmiVector * x )
```

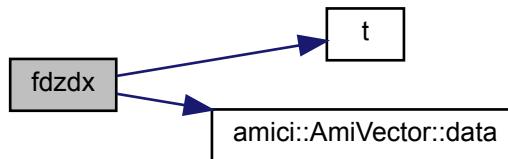
partial derivative of event-resolved output z w.r.t. to model states x

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state

Definition at line 885 of file model.cpp.

Here is the call graph for this function:



10.11.3.28 fdrzdp() [1/2]

```
void fdrzdp (
    const realltype t,
    const int ie,
    const AmiVector * x )
```

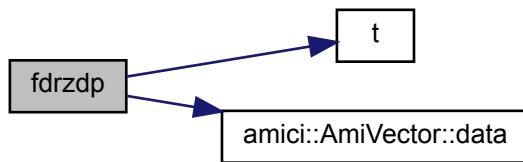
Sensitivity of event-resolved root output w.r.t. to model parameters p

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state

Definition at line 890 of file model.cpp.

Here is the call graph for this function:



10.11.3.29 fdrzdx() [1/2]

```
void fdrzdx (
    const realltype t,
    const int ie,
    const AmiVector * x )
```

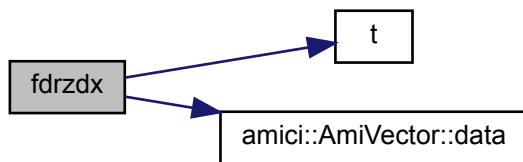
Sensitivity of event-resolved measurements rz w.r.t. to model states x

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state

Definition at line 898 of file model.cpp.

Here is the call graph for this function:



10.11.3.30 fdeltax() [1/2]

```
void fdeltax (
    const int ie,
    const realtype t,
    const AmiVector * x,
    const AmiVector * xdot,
    const AmiVector * xdot_old )
```

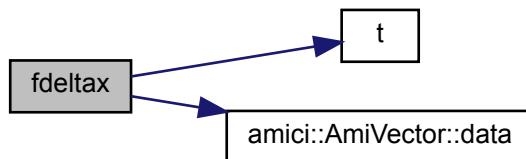
State update functions for events

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>xdot</i>	current residual function values
<i>xdot_old</i>	value of residual function before event

Definition at line 903 of file model.cpp.

Here is the call graph for this function:



10.11.3.31 fdeltasx() [1/2]

```
void fdeltasx (
    const int ie,
    const realtype t,
    const AmiVector * x,
    const AmiVectorArray * sx,
    const AmiVector * xdot,
    const AmiVector * xdot_old )
```

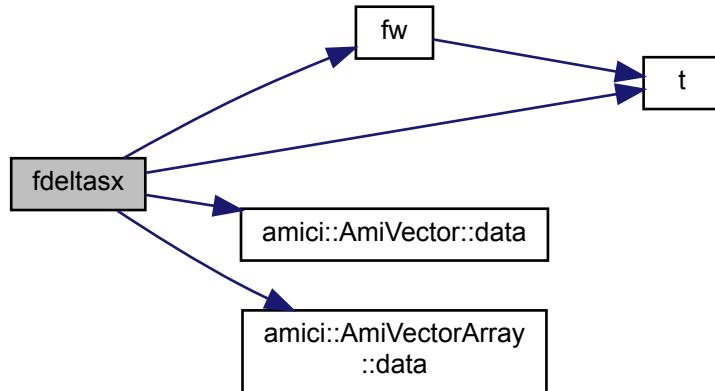
Sensitivity update functions for events, total derivative

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>sx</i>	current state sensitivity
<i>xdot</i>	current residual function values
<i>xdot_old</i>	value of residual function before event

Definition at line 909 of file model.cpp.

Here is the call graph for this function:



10.11.3.32 fdeltaxB() [1/2]

```
void fdeltaxB (
    const int ie,
    const realtype t,
    const AmiVector * x,
    const AmiVector * xB,
    const AmiVector * xdot,
    const AmiVector * xdot_old )
```

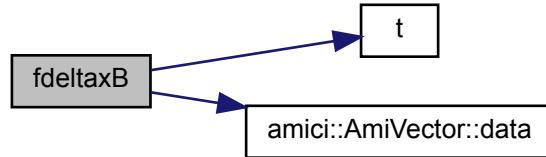
Adjoint state update functions for events

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>xB</i>	current adjoint state
<i>xdot</i>	current residual function values
<i>xdot_old</i>	value of residual function before event

Definition at line 918 of file model.cpp.

Here is the call graph for this function:



10.11.3.33 fdeltaqB() [1/2]

```
void fdeltaqB (
    const int ie,
    const realscale t,
    const AmiVector * x,
    const AmiVector * xB,
    const AmiVector * xdot,
    const AmiVector * xdot_old )
```

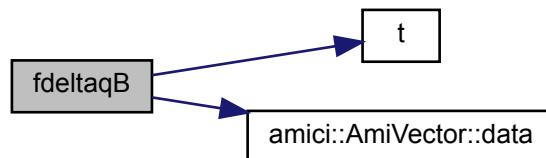
Quadrature state update functions for events

Parameters

<i>ie</i>	event index
<i>t</i>	current timepoint
<i>x</i>	current state
<i>xB</i>	current adjoint state
<i>xdot</i>	current residual function values
<i>xdot_old</i>	value of residual function before event

Definition at line 924 of file model.cpp.

Here is the call graph for this function:



10.11.3.34 **fsigmay()** [1/2]

```
void fsigmay (
    const int it,
    ReturnData * rdata,
    const ExpData * edata )
```

Standard deviation of measurements

Parameters

<i>it</i>	timepoint index
<i>edata</i>	pointer to experimental data instance
<i>rdata</i>	pointer to return data instance

Definition at line 932 of file model.cpp.

Here is the call graph for this function:

10.11.3.35 **fdsigmaydp()** [1/2]

```
void fdsigmaydp (
    const int it,
    ReturnData * rdata,
    const ExpData * edata )
```

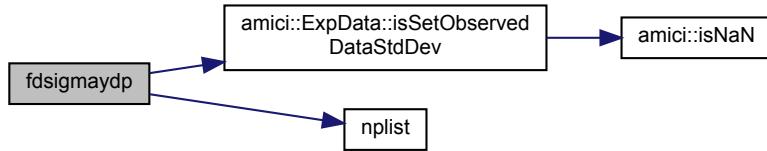
partial derivative of standard deviation of measurements w.r.t. model

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to <code>ExpData</code> data instance holding sigma values

Definition at line 951 of file model.cpp.

Here is the call graph for this function:



10.11.3.36 fsigmaz() [1/2]

```
void fsigmaz (
    const realltype t,
    const int ie,
    const int * nroots,
    ReturnData * rdata,
    const ExpData * edata )
```

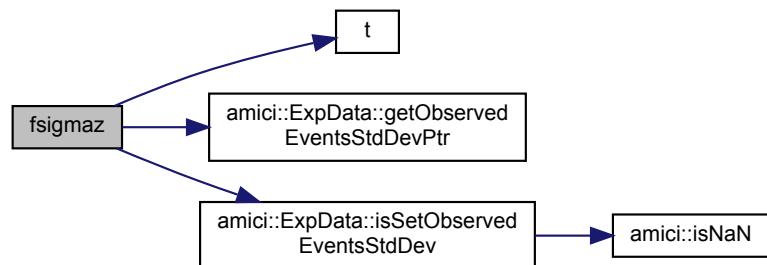
Standard deviation of events

Parameters

<code>t</code>	current timepoint
<code>ie</code>	event index
<code>nroots</code>	array with event numbers
<code>edata</code>	pointer to experimental data instance
<code>rdata</code>	pointer to return data instance

Definition at line 980 of file model.cpp.

Here is the call graph for this function:



10.11.3.37 `fdsigmazdp()` [1/2]

```
void fdsigmazdp (
    const realtype t,
    const int ie,
    const int * nroots,
    ReturnData * rdata,
    const ExpData * edata )
```

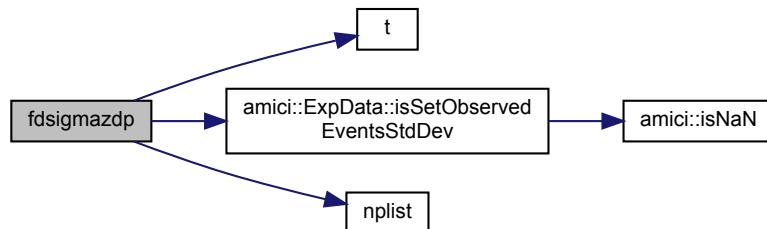
Sensitivity of standard deviation of events measurements w.r.t. model parameters p

Parameters

<i>t</i>	current timepoint
<i>ie</i>	event index
<i>nroots</i>	array with event numbers
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 998 of file model.cpp.

Here is the call graph for this function:

10.11.3.38 `fJy()` [1/2]

```
void fJy (
    const int it,
    ReturnData * rdata,
    const ExpData * edata )
```

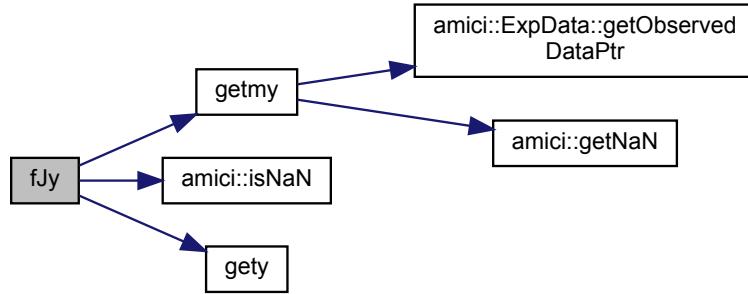
negative log-likelihood of measurements y

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1023 of file model.cpp.

Here is the call graph for this function:



10.11.3.39 fJz() [1/2]

```
void fJz (
    const int nroots,
    ReturnData * rdata,
    const ExpData * edata )
```

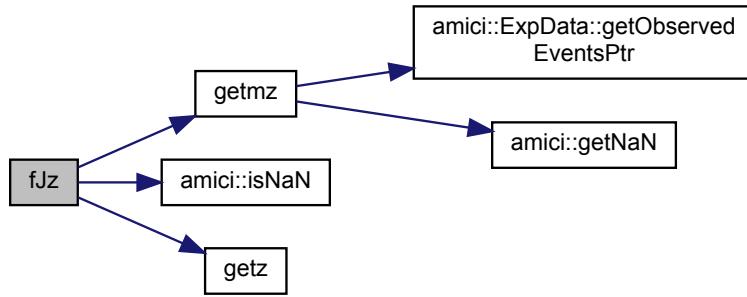
negative log-likelihood of event-resolved measurements z

Parameters

<i>nroots</i>	event index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1035 of file model.cpp.

Here is the call graph for this function:



10.11.3.40 fJrz() [1/2]

```
void fJrz (
    const int nroots,
    ReturnData * rdata,
    const ExpData * edata )
```

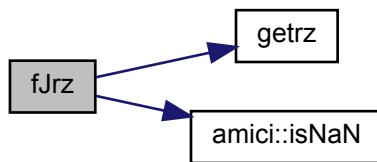
regularization of negative log-likelihood with roots of event-resolved measurements rz

Parameters

<i>nroots</i>	event index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1047 of file model.cpp.

Here is the call graph for this function:



10.11.3.41 fdJydy() [1/2]

```
void fdJydy (
    const int it,
    const ReturnData * rdata,
    const ExpData * edata )
```

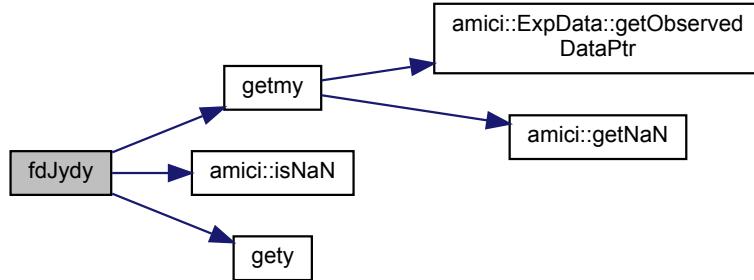
partial derivative of time-resolved measurement negative log-likelihood Jy

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1059 of file model.cpp.

Here is the call graph for this function:



10.11.3.42 fdJydsigma() [1/2]

```
void fdJydsigma (
    const int it,
    const ReturnData * rdata,
    const ExpData * edata )
```

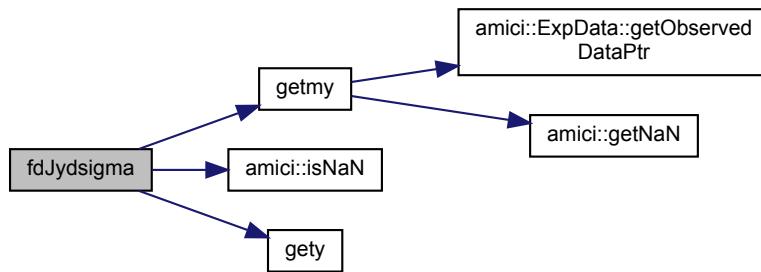
Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigma

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1079 of file model.cpp.

Here is the call graph for this function:



10.11.3.43 fdJzdz() [1/2]

```

void fdJzdz (
    const int nroots,
    const ReturnData * rdata,
    const ExpData * edata )
  
```

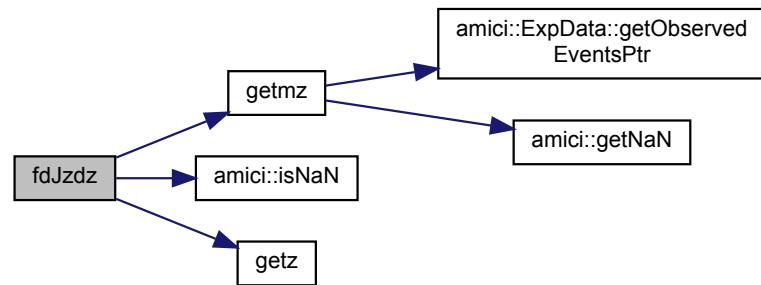
partial derivative of event measurement negative log-likelihood Jz

Parameters

<code>nroots</code>	event index
<code>rdata</code>	pointer to return data instance
<code>edata</code>	pointer to experimental data instance

Definition at line 1099 of file model.cpp.

Here is the call graph for this function:



10.11.3.44 fdJzdsigma() [1/2]

```
void fdJzdsigma (
    const int nroots,
    const ReturnData * rdata,
    const ExpData * edata )
```

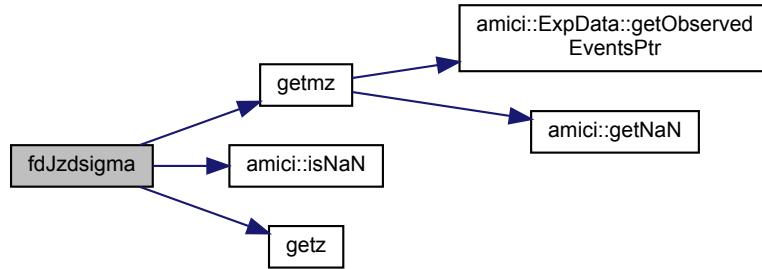
Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz

Parameters

<i>nroots</i>	event index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1110 of file model.cpp.

Here is the call graph for this function:



10.11.3.45 fdJrzdz() [1/2]

```
void fdJrzdz (
    const int nroots,
    const ReturnData * rdata,
    const ExpData * edata )
```

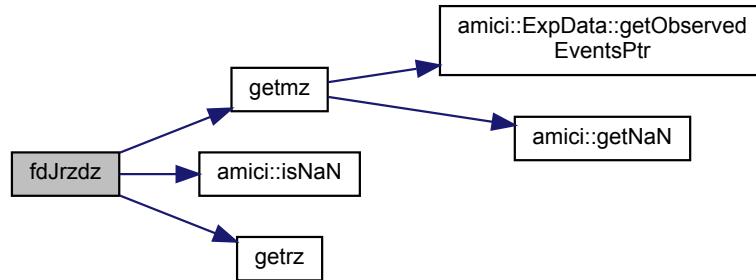
partial derivative of event measurement negative log-likelihood Jz

Parameters

<i>nroots</i>	event index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1121 of file model.cpp.

Here is the call graph for this function:



10.11.3.46 fdJrzdsigma() [1/2]

```
void fdJrzdsigma (
    const int nroots,
    const ReturnData * rdata,
    const ExpData * edata )
```

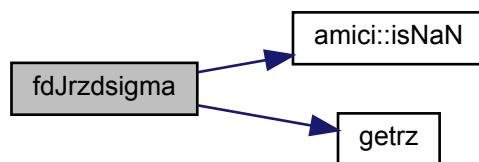
Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz

Parameters

<i>nroots</i>	event index
<i>rdata</i>	pointer to return data instance
<i>edata</i>	pointer to experimental data instance

Definition at line 1132 of file model.cpp.

Here is the call graph for this function:



10.11.3.47 fsy()

```
void fsy (
    const int it,
    ReturnData * rdata )
```

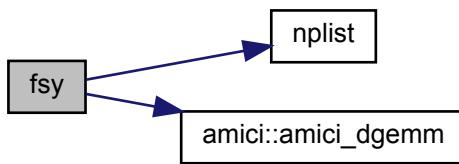
Sensitivity of measurements y, total derivative $sy = dydx * sx + dydp$

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Definition at line 14 of file model.cpp.

Here is the call graph for this function:



10.11.3.48 fsz_tf()

```
void fsz_tf (
    const int * nroots,
    const int ie,
    ReturnData * rdata )
```

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

Parameters

<i>nroots</i>	number of events for event index
<i>ie</i>	event index
<i>rdata</i>	pointer to return data instance

Definition at line 30 of file model.cpp.

Here is the call graph for this function:



10.11.3.49 fsJy()

```

void fsJy (
    const int it,
    const std::vector< realtype > & dJydx,
    ReturnData * rdata )
  
```

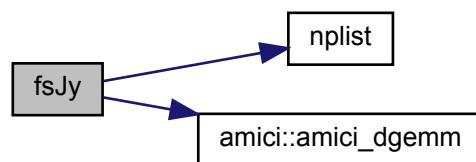
Sensitivity of time-resolved measurement negative log-likelihood Jy, total derivative

Parameters

<i>it</i>	timepoint index
<i>dJydx</i>	vector with values of state derivative of Jy
<i>rdata</i>	pointer to return data instance

Definition at line 38 of file model.cpp.

Here is the call graph for this function:



10.11.3.50 fdJydp()

```

void fdJydp (
    const int it,
  
```

```
const ExpData * edata,
ReturnData * rdata )
```

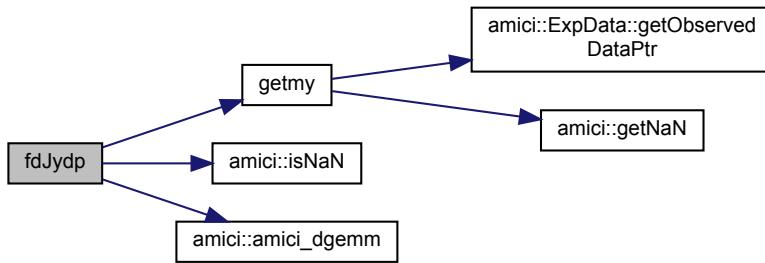
Compute sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. parameters for the given time-point. Add result to respective fields in rdata.

Parameters

<i>it</i>	timepoint index
<i>edata</i>	pointer to experimental data instance
<i>rdata</i>	pointer to return data instance

Definition at line 67 of file model.cpp.

Here is the call graph for this function:



10.11.3.51 fdJydx()

```
void fdJydx (
    std::vector< realtype > * dJydx,
    const int it,
    const ExpData * edata,
    const ReturnData * rdata )
```

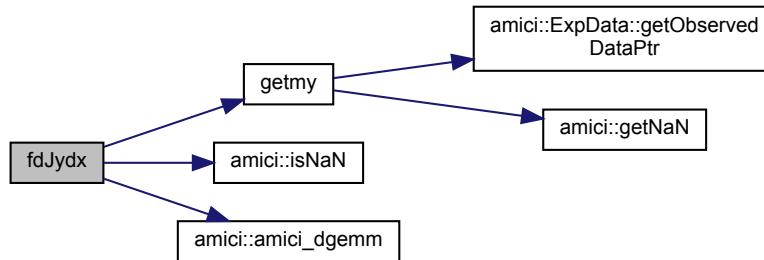
Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. state variables

Parameters

<i>dJydx</i>	pointer to vector with values of state derivative of Jy
<i>it</i>	timepoint index
<i>edata</i>	pointer to experimental data instance
<i>rdata</i>	pointer to return data instance

Definition at line 115 of file model.cpp.

Here is the call graph for this function:



10.11.3.52 fsJz()

```
void fsJz (
    const int nroots,
    const std::vector< realtype > & dJzdx,
    AmiVectorArray * sx,
    ReturnData * rdata )
```

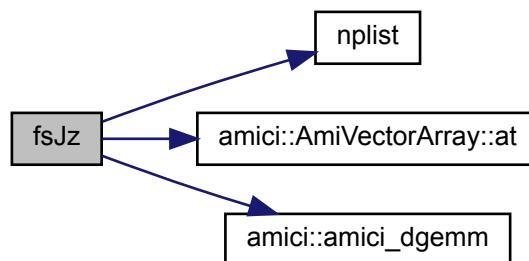
Sensitivity of event-resolved measurement negative log-likelihood Jz, total derivative

Parameters

<i>nroots</i>	event index
<i>dJzdx</i>	vector with values of state derivative of Jz
<i>sx</i>	pointer to state sensitivities
<i>rdata</i>	pointer to return data instance

Definition at line 134 of file model.cpp.

Here is the call graph for this function:



10.11.3.53 fdJzdp()

```
void fdJzdp (
    const int nroots,
    realtype t,
    const ExpData * edata,
    const ReturnData * rdata )
```

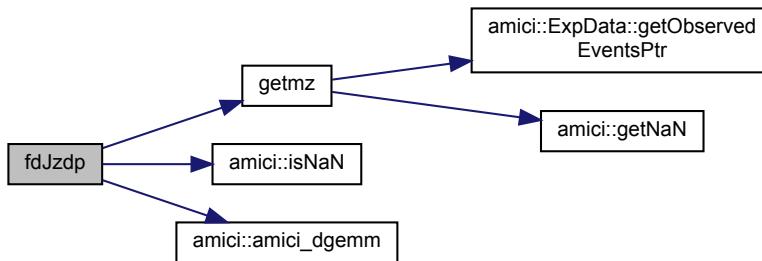
Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. parameters

Parameters

<i>nroots</i>	event index
<i>t</i>	current timepoint
<i>edata</i>	pointer to experimental data instance
<i>rdata</i>	pointer to return data instance

Definition at line 168 of file model.cpp.

Here is the call graph for this function:



10.11.3.54 fdJzdx()

```
void fdJzdx (
    std::vector< realtype > * dJzdx,
    const int nroots,
    realtype t,
    const ExpData * edata,
    const ReturnData * rdata )
```

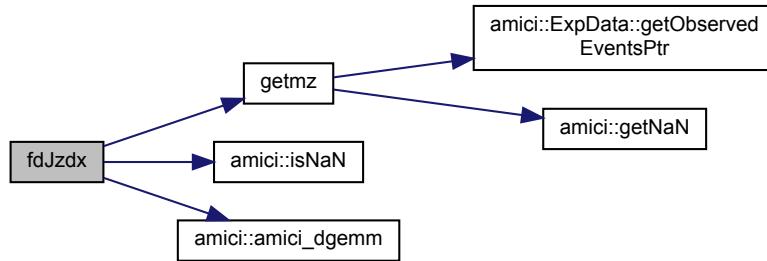
Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. state variables

Parameters

<i>dJzdx</i>	pointer to vector with values of state derivative of Jz
<i>nroots</i>	event index
<i>t</i>	current timepoint
<i>edata</i>	pointer to experimental data instance
<i>rdata</i>	pointer to return data instance

Definition at line 204 of file model.cpp.

Here is the call graph for this function:



10.11.3.55 initialize()

```
void initialize (
    AmiVector * x,
    AmiVector * dx )
```

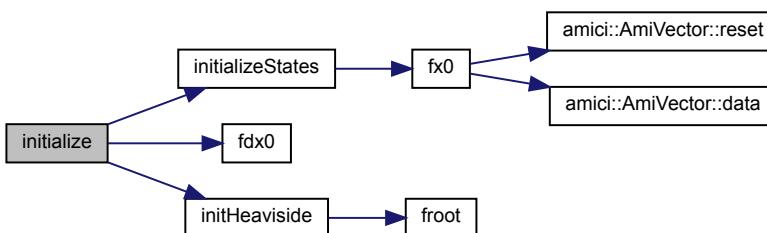
initialization of model properties

Parameters

<i>x</i>	pointer to state variables
<i>dx</i>	pointer to time derivative of states (DAE only)

Definition at line 227 of file model.cpp.

Here is the call graph for this function:



10.11.3.56 initializeStates()

```
void initializeStates (
    AmiVector * x )
```

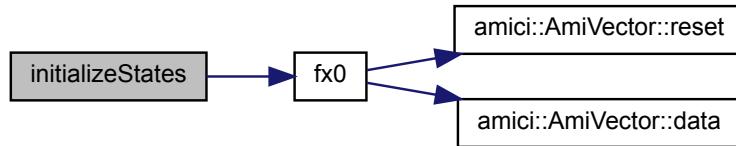
initialization of initial states

Parameters

<code>x</code>	pointer to state variables
----------------	----------------------------

Definition at line 238 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.57 initHeaviside()

```
void initHeaviside (
    AmiVector * x,
    AmiVector * dx )
```

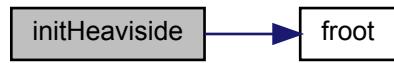
`initHeaviside` initialises the heaviside variables `h` at the intial time `t0` heaviside variables activate/deactivate on event occurrences

Parameters

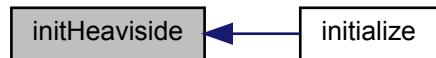
<code>x</code>	pointer to state variables
<code>dx</code>	pointer to time derivative of states (DAE only)

Definition at line 249 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.58 nplist()

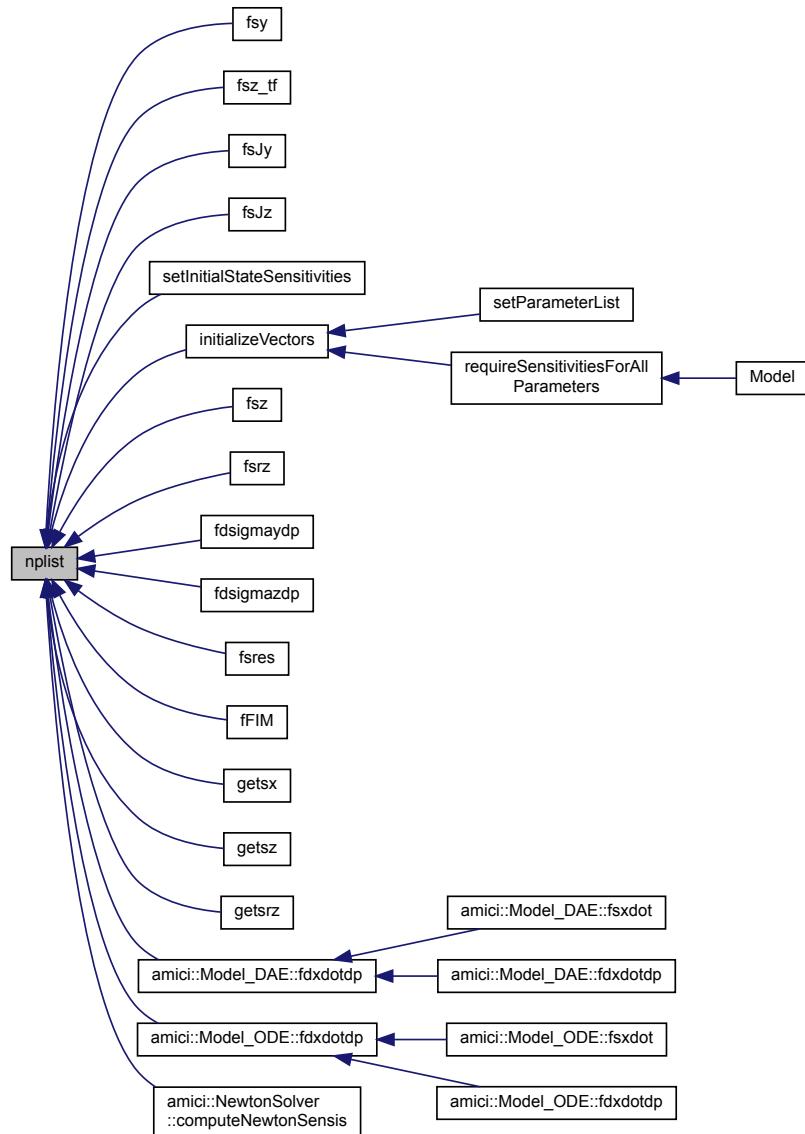
```
int nplist ( ) const
```

Returns

length of sensitivity index vector

Definition at line 266 of file model.cpp.

Here is the caller graph for this function:



10.11.3.59 np()

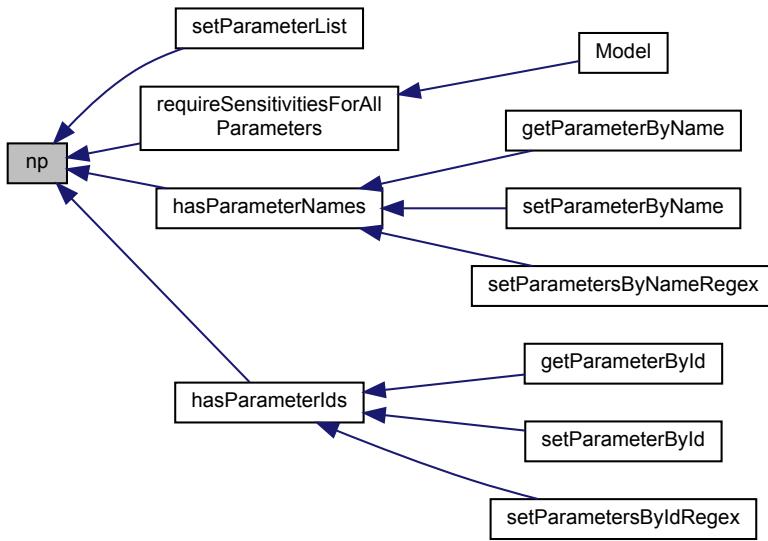
```
int np( ) const
```

Returns

length of parameter vector

Definition at line 270 of file model.cpp.

Here is the caller graph for this function:



10.11.3.60 nk()

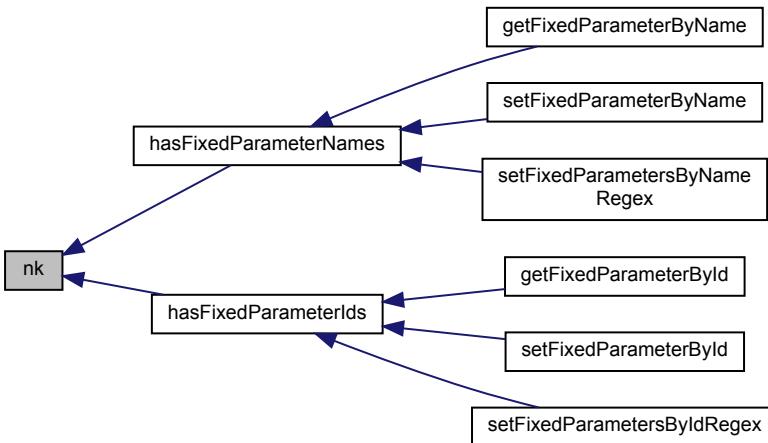
```
int nk( ) const
```

Returns

length of constant vector

Definition at line 274 of file model.cpp.

Here is the caller graph for this function:



10.11.3.61 k()

```
const double * k ( ) const
```

Returns

pointer to constants array

Definition at line 278 of file model.cpp.

Here is the caller graph for this function:



10.11.3.62 nMaxEvent()

```
int nMaxEvent ( ) const
```

Returns

maximum number of events that may occur for each type

Definition at line 282 of file model.cpp.

10.11.3.63 setNMaxEvent()

```
void setNMaxEvent ( int nmaxevent )
```

Parameters

<i>nmaxevent</i>	maximum number of events that may occur for each type
------------------	---

Definition at line 286 of file model.cpp.

10.11.3.64 nt()

```
int nt ( ) const
```

Returns

number of timepoints

Definition at line 290 of file model.cpp.

10.11.3.65 getParameterScale()

```
const std::vector< ParameterScaling > & getParameterScale ( ) const
```

Returns

vector of parameter scale

Definition at line 294 of file model.cpp.

10.11.3.66 setParameterScale() [1/2]

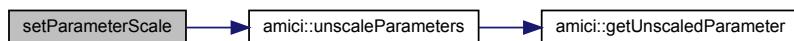
```
void setParameterScale (
    ParameterScaling pscale )
```

Parameters

<i>pscale</i>	scalar parameter scale for all parameters
---------------	---

Definition at line 298 of file model.cpp.

Here is the call graph for this function:

**10.11.3.67 setParameterScale() [2/2]**

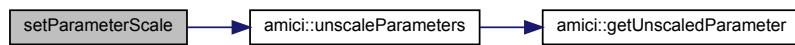
```
void setParameterScale (
    const std::vector< ParameterScaling > & pscale )
```

Parameters

<i>pscale</i>	vector of parameter scales
---------------	----------------------------

Definition at line 304 of file model.cpp.

Here is the call graph for this function:

**10.11.3.68 getParameters()**

```
std::vector< realtypes > const & getParameters ( ) const
```

Returns

The user-set parameters (see also getUnscaledParameters)

Definition at line 310 of file model.cpp.

10.11.3.69 setParameters()

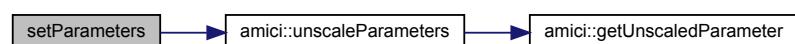
```
void setParameters (
    std::vector< realtypes > const & p )
```

Parameters

<i>p</i>	vector of parameters
----------	----------------------

Definition at line 397 of file model.cpp.

Here is the call graph for this function:



10.11.3.70 getUnscaledParameters()

```
const std::vector< realtypes > & getUnscaledParameters ( ) const
```

Returns

unscaled parameters

Definition at line 459 of file model.cpp.

10.11.3.71 getFixedParameters()

```
const std::vector< realtypes > & getFixedParameters ( ) const
```

Returns

vector of fixed parameters

Definition at line 463 of file model.cpp.

10.11.3.72 setFixedParameters()

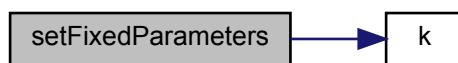
```
void setFixedParameters (   
    std::vector< realtypes > const & k )
```

Parameters

<i>k</i>	vector of fixed parameters
----------	----------------------------

Definition at line 489 of file model.cpp.

Here is the call graph for this function:

**10.11.3.73 getFixedParameterById()**

```
realtypes getFixedParameterById (   
    std::string const & par_id ) const
```

Parameters

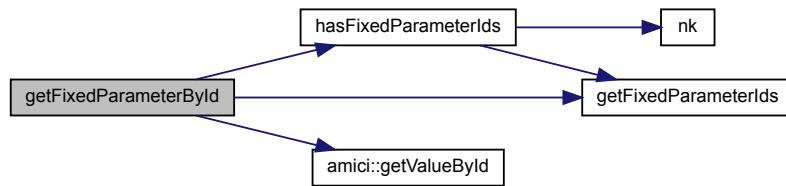
<i>par</i> <i>_id</i>	parameter id
--------------------------	--------------

Returns

parameter value

Definition at line 467 of file model.cpp.

Here is the call graph for this function:

**10.11.3.74 getFixedParameterByName()**

```

realtype getFixedParameterByName (
    std::string const & par_name ) const
  
```

Parameters

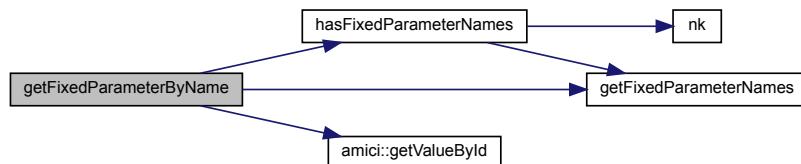
<i>par_name</i>	parameter name
-----------------	----------------

Returns

parameter value

Definition at line 478 of file model.cpp.

Here is the call graph for this function:



10.11.3.75 setFixedParameterById()

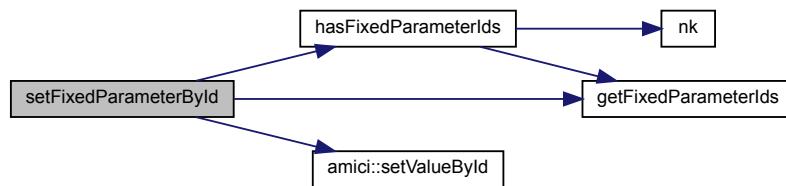
```
void setFixedParameterById (
    std::string const & par_id,
    realtype value )
```

Parameters

<i>par_id</i>	fixed parameter id
<i>value</i>	fixed parameter value

Definition at line 495 of file model.cpp.

Here is the call graph for this function:



10.11.3.76 setFixedParametersByIdRegex()

```
int setFixedParametersByIdRegex (
    std::string const & par_id_regex,
    realtype value )
```

Parameters

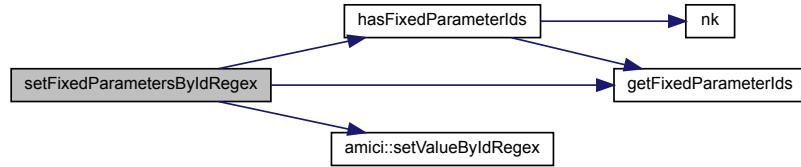
<i>par_id_regex</i>	fixed parameter name regex
<i>value</i>	fixed parameter value

Returns

number of fixed parameter ids that matched the regex

Definition at line 507 of file model.cpp.

Here is the call graph for this function:



10.11.3.77 setFixedParameterByName()

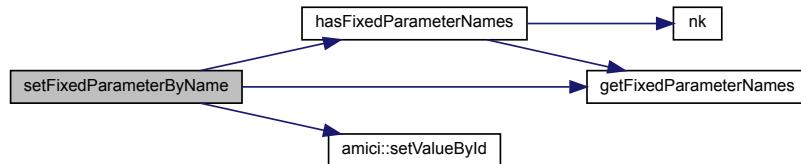
```
void setFixedParameterByName (
    std::string const & par_name,
    realtype value )
```

Parameters

<code>par_name</code>	fixed parameter id
<code>value</code>	fixed parameter value

Definition at line 519 of file `model.cpp`.

Here is the call graph for this function:



10.11.3.78 setFixedParametersByNameRegex()

```
int setFixedParametersByNameRegex (
    std::string const & par_name_regex,
    realtype value )
```

Parameters

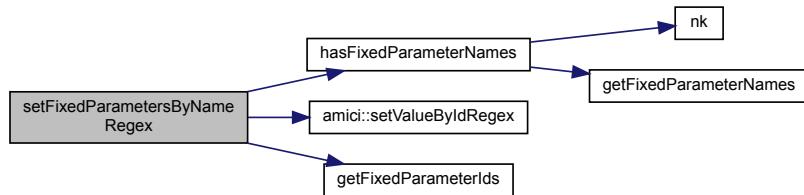
<code>par_name_regex</code>	fixed parameter name regex
<code>value</code>	fixed parameter value

Returns

number of fixed parameter names that matched the regex

Definition at line 531 of file model.cpp.

Here is the call graph for this function:

**10.11.3.79 getTimepoints()**

```
std::vector< realtyp > const & getTimepoints ( ) const
```

Returns

timepoint vector

Definition at line 543 of file model.cpp.

10.11.3.80 setTimepoints()

```
void setTimepoints (
    std::vector< realtyp > const & ts )
```

Parameters

<i>ts</i>	timepoint vector
-----------	------------------

Definition at line 547 of file model.cpp.

10.11.3.81 getStateIsNonNegative()

```
std::vector< bool > const & getStateIsNonNegative ( ) const
```

Returns

vector of flags

Definition at line 553 of file model.cpp.

10.11.3.82 setStateIsNonNegative()

```
void setStateIsNonNegative (
    std::vector< bool > const & stateIsNonNegative )
```

Parameters

<i>stateIsNonNegative</i>	vector of flags
---------------------------	-----------------

Definition at line 557 of file model.cpp.

10.11.3.83 t()

```
double t (
    int idx ) const
```

Parameters

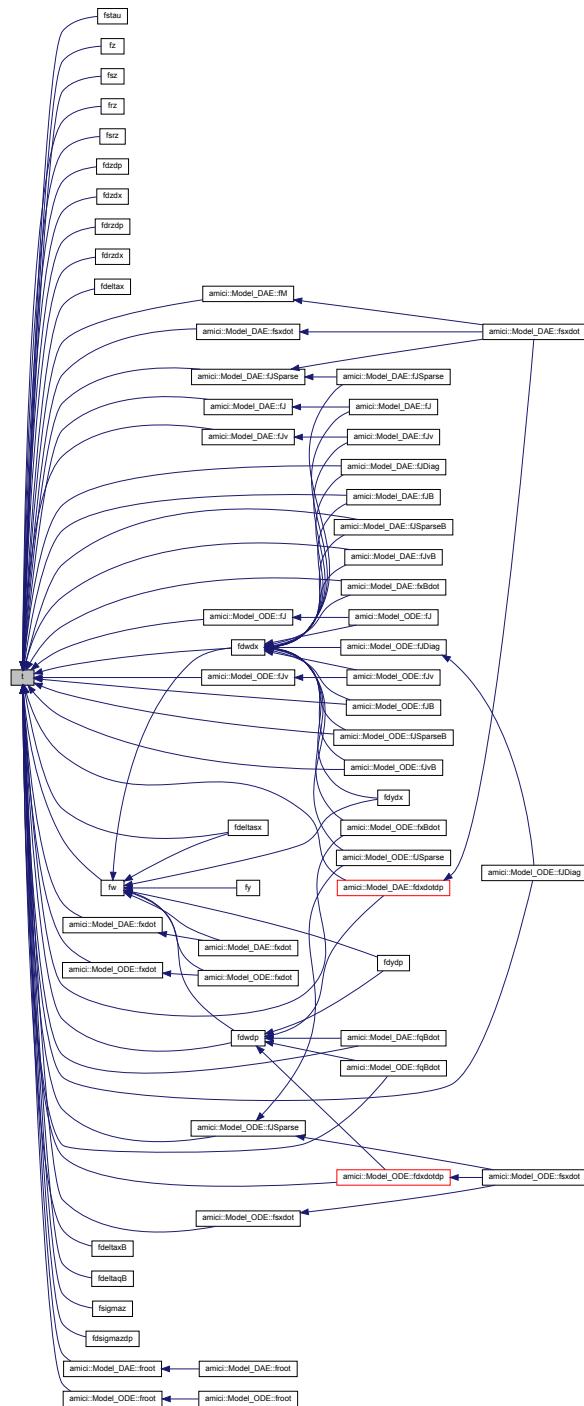
<i>idx</i>	timepoint index
------------	-----------------

Returns

timepoint

Definition at line 570 of file model.cpp.

Here is the caller graph for this function:



10.11.3.84 getParameterList()

```
const std::vector< int > & getParameterList ( ) const
```

Returns

list of parameter indices

Definition at line 574 of file model.cpp.

10.11.3.85 setParameterList()

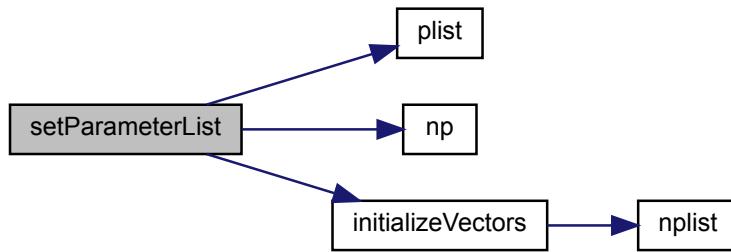
```
void setParameterList (
    std::vector< int > const & plist )
```

Parameters

<i>plist</i>	list of parameter indices
--------------	---------------------------

Definition at line 578 of file model.cpp.

Here is the call graph for this function:

**10.11.3.86 getInitialStates()**

```
std::vector< realltype > const & getInitialStates ( ) const
```

Returns

initial state vector

Definition at line 587 of file model.cpp.

10.11.3.87 setInitialStates()

```
void setInitialStates (
    std::vector< realltype > const & x0 )
```

Parameters

x0	initial state vector
----	----------------------

Definition at line 591 of file model.cpp.

10.11.3.88 getInitialStateSensitivities()

```
const std::vector< realtypes > & getInitialStateSensitivities ( ) const
```

Returns

vector of initial state sensitivities

Definition at line 600 of file model.cpp.

10.11.3.89 setInitialStateSensitivities()

```
void setInitialStateSensitivities (
    std::vector< realtypes > const & sx0 )
```

Parameters

sx0	vector of initial state sensitivities
-----	---------------------------------------

Definition at line 604 of file model.cpp.

Here is the call graph for this function:

**10.11.3.90 t0()**

```
double t0 ( ) const
```

Returns

simulation start time

Definition at line 613 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.91 setT0()**

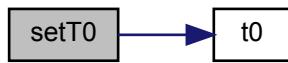
```
void setT0 ( double t0 )
```

Parameters

t0	simulation start time
----	-----------------------

Definition at line 617 of file model.cpp.

Here is the call graph for this function:

**10.11.3.92 plist()**

```
int plist ( int pos ) const
```

Parameters

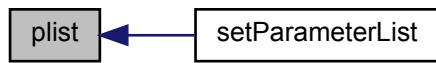
pos	index
-----	-------

Returns

entry

Definition at line 621 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.93 fw() [1/2]**

```
void fw (
    const realltype t,
    const realltype * x )
```

Parameters

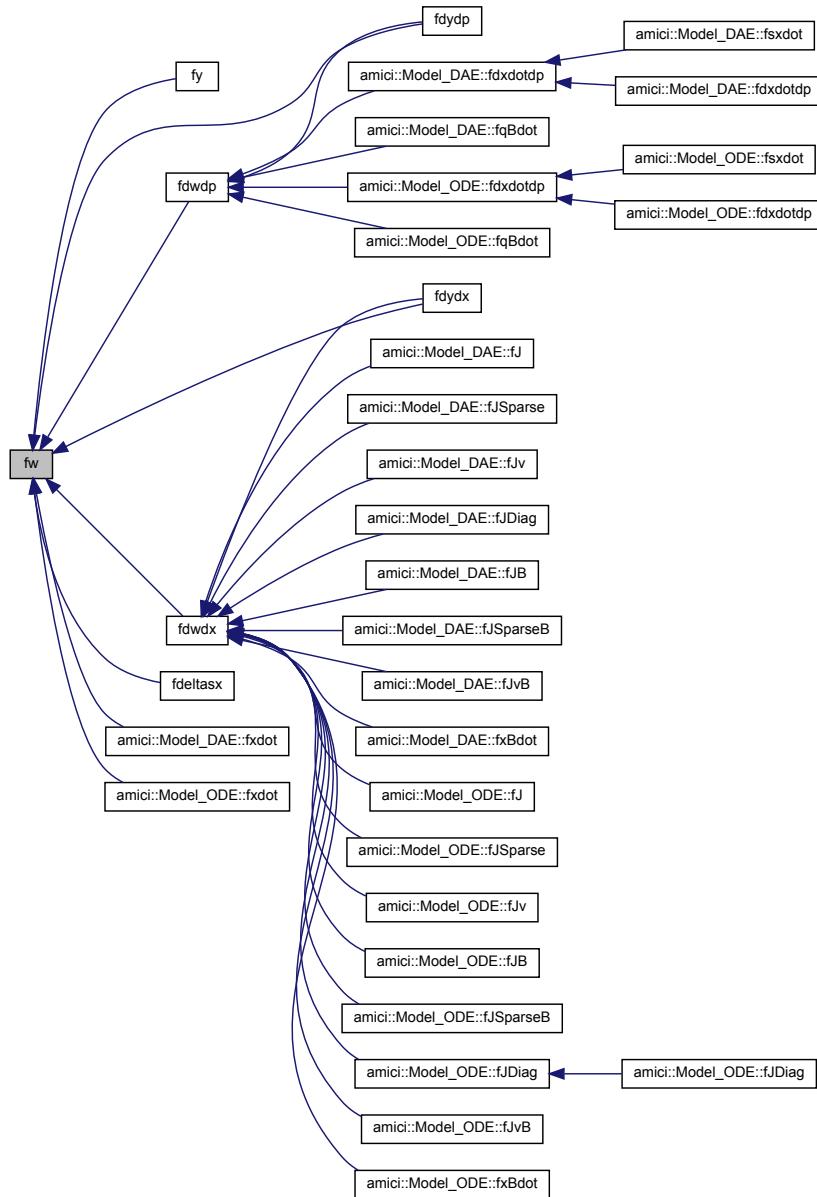
<i>t</i>	timepoint
<i>x</i>	array with the states

Definition at line 1142 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.94 fdwdp() [1/2]

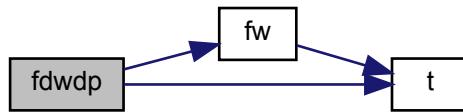
```
void fdwdp ( const realtype t, const realtype * x )
```

Parameters

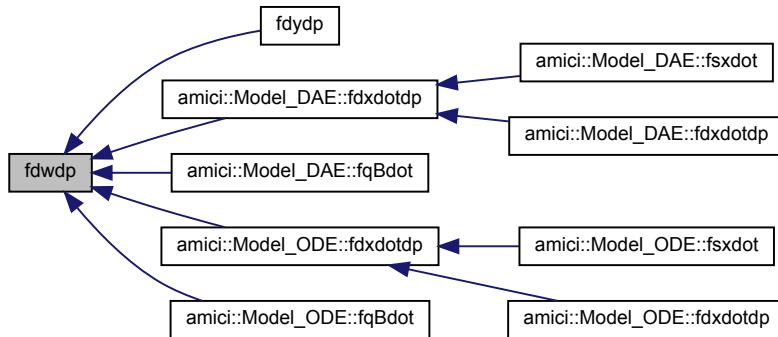
t	timepoint
x	array with the states

Definition at line 1147 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.95 fdwdx() [1/2]

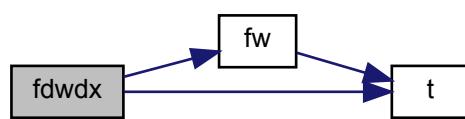
```
void fdwdx (
    const realltype t,
    const realltype * x )
```

Parameters

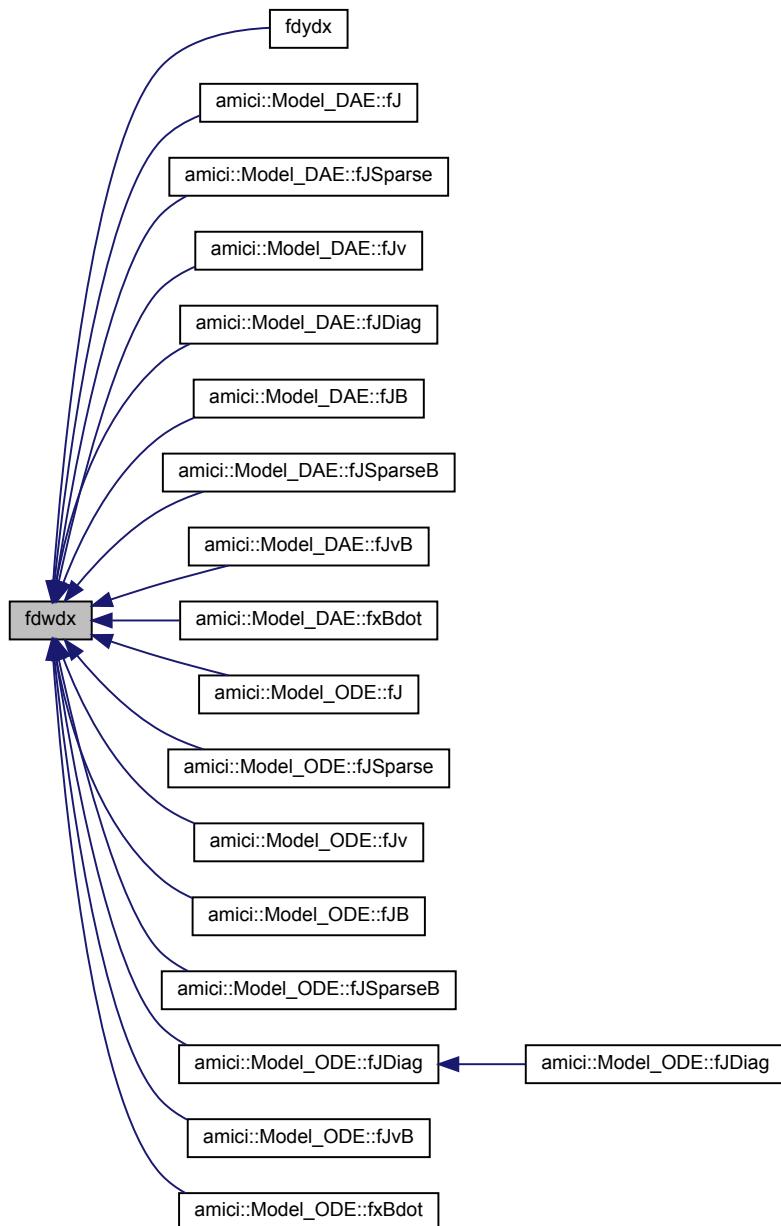
<i>t</i>	timepoint
<i>x</i>	array with the states

Definition at line 1153 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.96 `fres()`

```
void fres (
    const int it,
    ReturnData * rdata,
    const ExpData * edata )
```

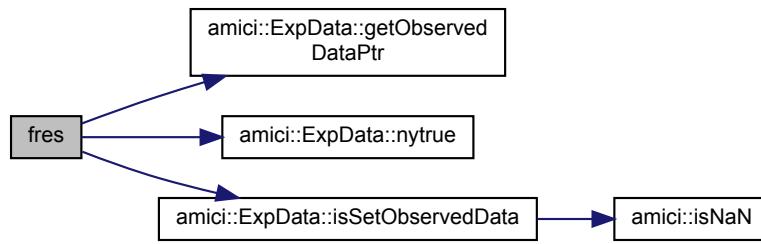
residual function

Parameters

<i>it</i>	time index
<i>rdata</i>	ReturnData instance to which result will be written
<i>edata</i>	ExpData instance containing observable data

Definition at line 1159 of file model.cpp.

Here is the call graph for this function:

**10.11.3.97 fchi2()**

```
void fchi2 (
    const int it,
    ReturnData * rdata )
```

chi-squared function

Parameters

<i>it</i>	time index
<i>rdata</i>	ReturnData instance to which result will be written

Definition at line 1174 of file model.cpp.

10.11.3.98 fsres()

```
void fsres (
    const int it,
    ReturnData * rdata,
    const ExpData * edata )
```

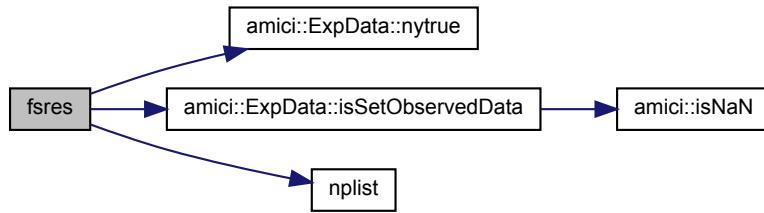
residual sensitivity function

Parameters

<i>it</i>	time index
<i>rdata</i>	ReturnData instance to which result will be written
<i>edata</i>	ExpData instance containing observable data

Definition at line 1184 of file model.cpp.

Here is the call graph for this function:

**10.11.3.99 fFIM()**

```
void fFIM (
    const int it,
    ReturnData * rdata )
```

fisher information matrix function

Parameters

<i>it</i>	time index
<i>rdata</i>	ReturnData instance to which result will be written

Definition at line 1200 of file model.cpp.

Here is the call graph for this function:



10.11.3.100 updateHeaviside()

```
void updateHeaviside (
    const std::vector< int > & rootsfound )
```

updateHeaviside updates the heaviside variables h on event occurrences

Parameters

<i>rootsfound</i>	provides the direction of the zero-crossing, so adding it will give the right update to the heaviside variables (zero if no root was found)
-------------------	---

Definition at line 1216 of file model.cpp.

10.11.3.101 updateHeavisideB()

```
void updateHeavisideB (
    const int * rootsfound )
```

updateHeavisideB updates the heaviside variables h on event occurrences in the backward problem

Parameters

<i>rootsfound</i>	provides the direction of the zero-crossing, so adding it will give the right update to the heaviside variables (zero if no root was found)
-------------------	---

Definition at line 1222 of file model.cpp.

10.11.3.102 gett()

```
realtype gett (
    const int it,
    const ReturnData * rdata ) const
```

get current timepoint from index

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Returns

current timepoint

Definition at line 1249 of file model.cpp.

10.11.3.103 checkFinite()

```
int checkFinite (
    const int N,
    const realtype * array,
    const char * fun ) const
```

Parameters

<i>N</i>	number of datapoints in array
<i>array</i>	arrays of values
<i>fun</i>	name of the function that generated the values

Returns

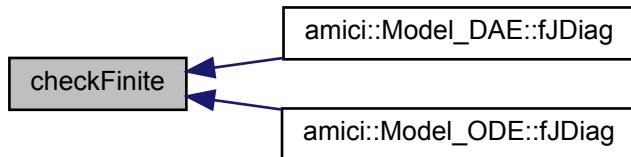
AMICI_RECOVERABLE_ERROR if a NaN/Inf value was found, AMICI_SUCCESS otherwise

Definition at line 1277 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.104 hasParameterNames()

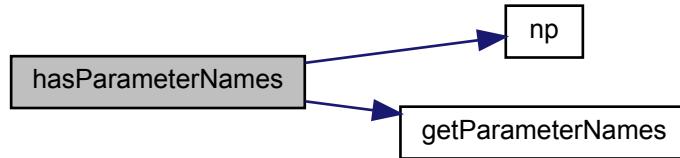
```
virtual bool hasParameterNames ( ) const [virtual]
```

Returns

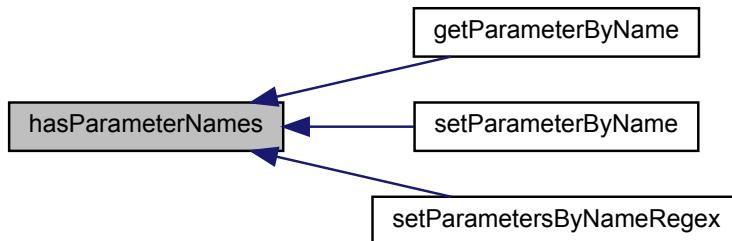
boolean indicating whether parameter names were set

Definition at line 881 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.105 getParameterNames()

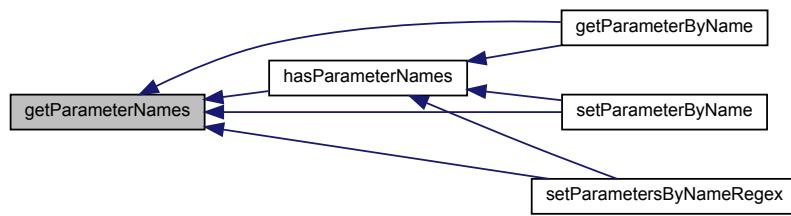
```
virtual std::vector<std::string> getParameterNames ( ) const [virtual]
```

Returns

the names

Definition at line 887 of file model.h.

Here is the caller graph for this function:

**10.11.3.106 hasStateNames()**

```
virtual bool hasStateNames( ) const [virtual]
```

Returns

boolean indicating whether state names were set

Definition at line 893 of file model.h.

Here is the call graph for this function:



10.11.3.107 getStateNames()

```
virtual std::vector<std::string> getStateNames ( ) const [virtual]
```

Returns

the names

Definition at line 899 of file model.h.

Here is the caller graph for this function:

**10.11.3.108 hasFixedParameterNames()**

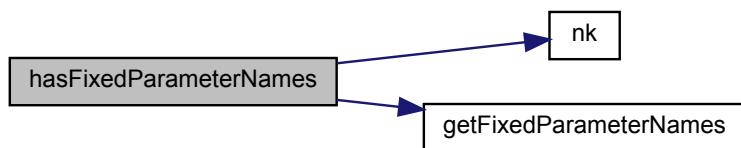
```
virtual bool hasFixedParameterNames ( ) const [virtual]
```

Returns

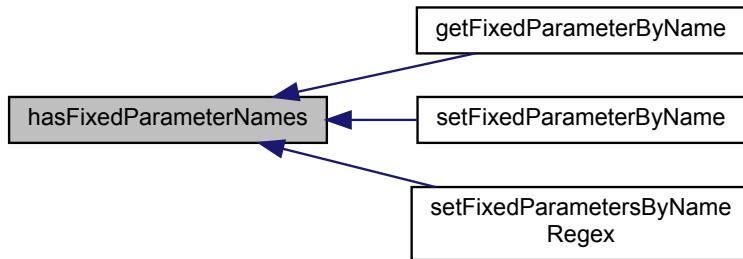
boolean indicating whether fixed parameter names were set

Definition at line 905 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.109 `getFixedParameterNames()`

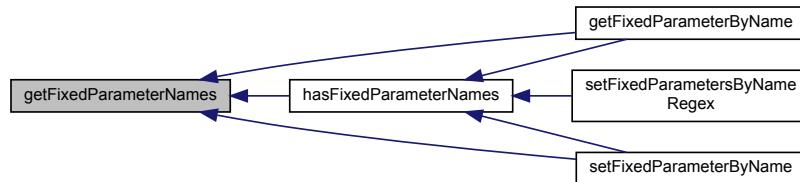
```
virtual std::vector<std::string> getFixedParameterNames( ) const [virtual]
```

Returns

the names

Definition at line 911 of file model.h.

Here is the caller graph for this function:



10.11.3.110 `hasObservableNames()`

```
virtual bool hasObservableNames( ) const [virtual]
```

Returns

boolean indicating whether observable names were set

Definition at line 917 of file model.h.

Here is the call graph for this function:

**10.11.3.111 getObservableNames()**

```
virtual std::vector<std::string> getObservableNames( ) const [virtual]
```

Returns

the names

Definition at line 923 of file model.h.

Here is the caller graph for this function:

**10.11.3.112 hasParameterIds()**

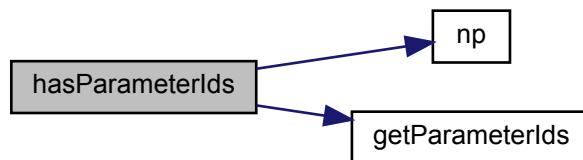
```
virtual bool hasParameterIds( ) const [virtual]
```

Returns

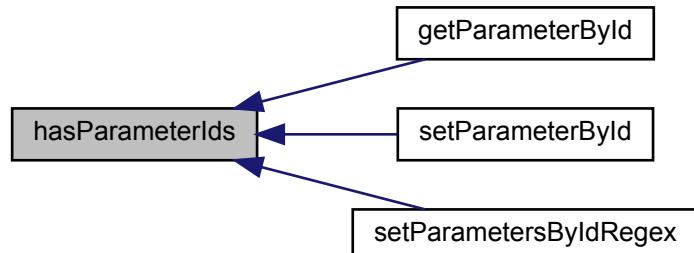
boolean indicating whether parameter ids were set

Definition at line 929 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.113 getParameterIds()**

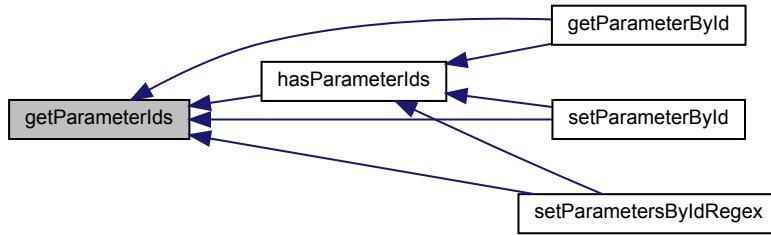
```
virtual std::vector<std::string> getParameterIds ( ) const [virtual]
```

Returns

the ids

Definition at line 935 of file model.h.

Here is the caller graph for this function:



10.11.3.114 getParameterById()

```
realtype getParameterById (
    std::string const & par_id ) const
```

Parameters

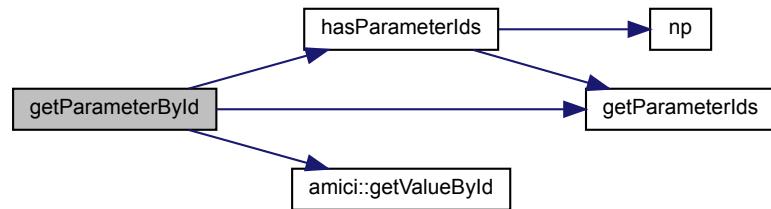
<i>par_id</i>	parameter id
---------------	--------------

Returns

parameter value

Definition at line 381 of file model.cpp.

Here is the call graph for this function:



10.11.3.115 getParameterByName()

```
realtype getParameterByName (
    std::string const & par_name ) const
```

Parameters

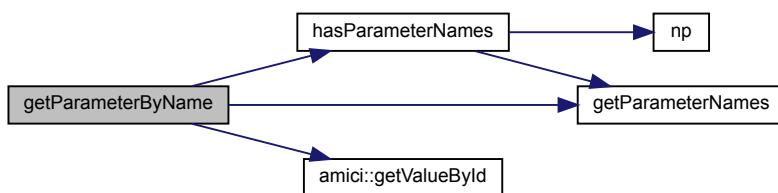
<i>par_name</i>	parameter name
-----------------	----------------

Returns

parameter value

Definition at line 387 of file model.cpp.

Here is the call graph for this function:

**10.11.3.116 setParameterById()**

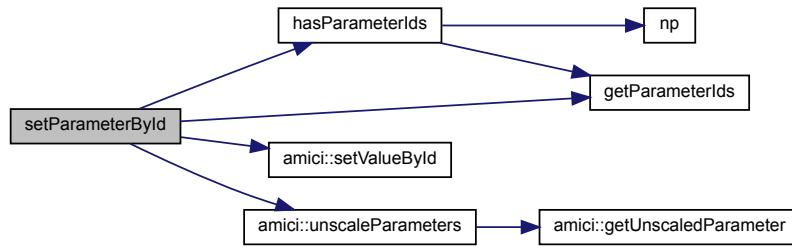
```
void setParameterById (
    std::string const & par_id,
    realtype value )
```

Parameters

<i>par_id</i>	parameter id
<i>value</i>	parameter value

Definition at line 405 of file model.cpp.

Here is the call graph for this function:



10.11.3.117 setParametersByIdRegex()

```
int setParametersByIdRegex (
    std::string const & par_id_regex,
    realtype value )
```

Parameters

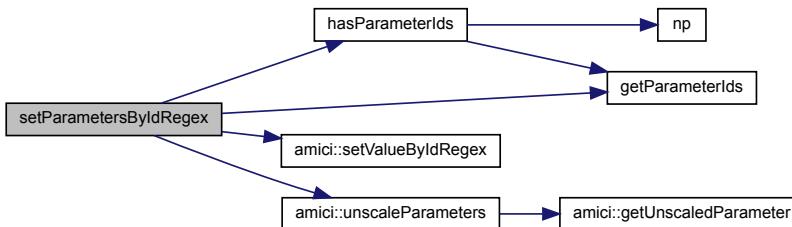
<i>par_id_regex</i>	parameter id regex
<i>value</i>	parameter value

Returns

number of parameter ids that matched the regex

Definition at line 418 of file model.cpp.

Here is the call graph for this function:



10.11.3.118 setParameterByName()

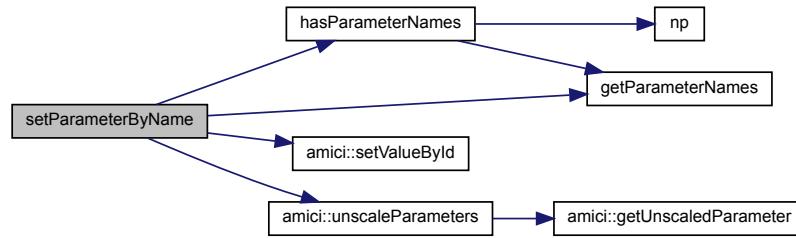
```
void setParameterByName (
    std::string const & par_name,
    realtype value )
```

Parameters

<i>par_name</i>	parameter name
<i>value</i>	parameter value

Definition at line 431 of file model.cpp.

Here is the call graph for this function:

**10.11.3.119 setParametersByNameRegex()**

```
int setParametersByNameRegex (
    std::string const & par_name_regex,
    realtype value )
```

Parameters

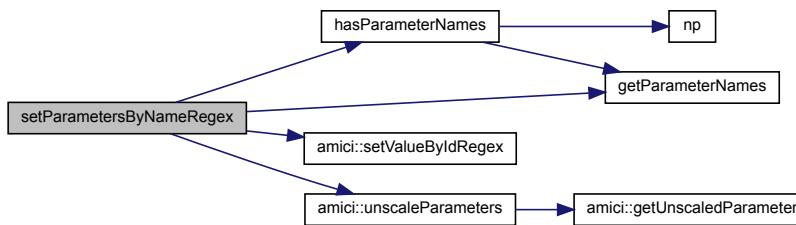
<i>par_name_regex</i>	parameter name regex
<i>value</i>	parameter value

Returns

number of fixed parameter names that matched the regex

Definition at line 444 of file model.cpp.

Here is the call graph for this function:



10.11.3.120 hasStateIds()

```
virtual bool hasStateIds ( ) const [virtual]
```

Returns

Definition at line 987 of file model.h.

Here is the call graph for this function:

**10.11.3.121 getStateIds()**

```
virtual std::vector<std::string> getStateIds ( ) const [virtual]
```

Returns

the ids

Definition at line 993 of file model.h.

Here is the caller graph for this function:



10.11.3.122 hasFixedParameterIds()

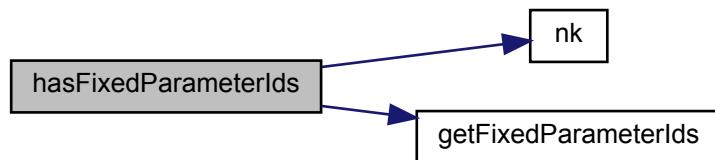
```
virtual bool hasFixedParameterIds () const [virtual]
```

Returns

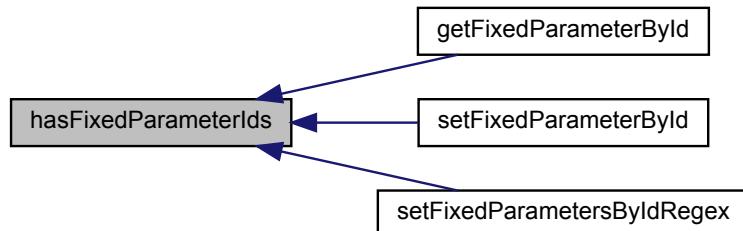
boolean indicating whether fixed parameter ids were set

Definition at line 1001 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.123 getFixedParameterIds()**

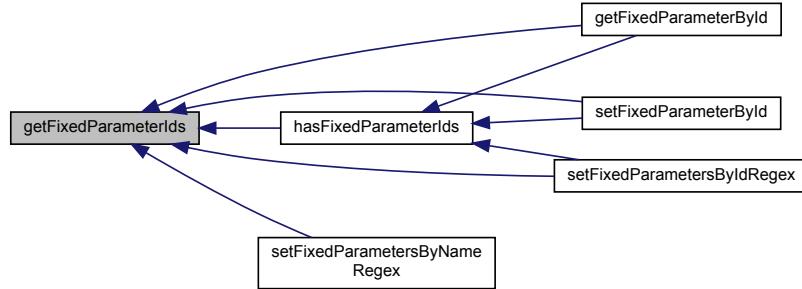
```
virtual std::vector<std::string> getFixedParameterIds () const [virtual]
```

Returns

the ids

Definition at line 1007 of file model.h.

Here is the caller graph for this function:

**10.11.3.124 hasObservableIds()**

```
virtual bool hasObservableIds ( ) const [virtual]
```

Returns

boolean indicating whether observale ids were set

Definition at line 1015 of file model.h.

Here is the call graph for this function:



10.11.3.125 getObservableIds()

```
virtual std::vector<std::string> getObservableIds() const [virtual]
```

Returns

the ids

Definition at line 1021 of file model.h.

Here is the caller graph for this function:

**10.11.3.126 setSteadyStateSensitivityMode()**

```
void setSteadyStateSensitivityMode( const SteadyStateSensitivityMode mode )
```

Parameters

<i>mode</i>	steadyStateSensitivityMode
-------------	----------------------------

Definition at line 1029 of file model.h.

10.11.3.127 getSteadyStateSensitivityMode()

```
SteadyStateSensitivityMode getSteadyStateSensitivityMode() const
```

Returns

flag value

Definition at line 1037 of file model.h.

10.11.3.128 setReinitializeFixedParameterInitialStates()

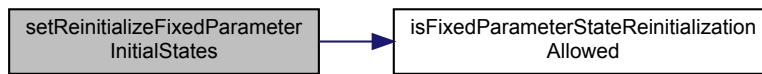
```
void setReinitializeFixedParameterInitialStates( bool flag )
```

Parameters

<i>flag</i>	true/false
-------------	------------

Definition at line 1046 of file model.h.

Here is the call graph for this function:

**10.11.3.129 getReinitializeFixedParameterInitialStates()**

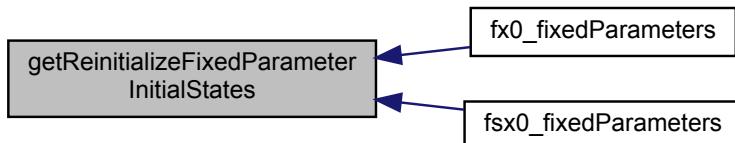
```
bool getReinitializeFixedParameterInitialStates ( ) const
```

Returns

flag true/false

Definition at line 1060 of file model.h.

Here is the caller graph for this function:

**10.11.3.130 fx0() [2/2]**

```
virtual void fx0 (
    realtype * x0,
    const realtype t,
    const realtype * p,
    const realtype * k ) [protected], [virtual]
```

model specific implementation of fx0

Parameters

<i>x0</i>	initial state
<i>t</i>	initial time
<i>p</i>	parameter vector
<i>k</i>	constant vector

Definition at line 1140 of file model.h.

10.11.3.131 fx0_fixedParameters() [2/2]

```
virtual void fx0_fixedParameters (
    realtype * x0,
    const realtype t,
    const realtype * p,
    const realtype * k ) [protected], [virtual]
```

model specific implementation of fx0_fixedParameters

Parameters

<i>x0</i>	initial state
<i>t</i>	initial time
<i>p</i>	parameter vector
<i>k</i>	constant vector

Definition at line 1150 of file model.h.

10.11.3.132 fsx0_fixedParameters() [2/2]

```
virtual void fsx0_fixedParameters (
    realtype * sx0,
    const realtype t,
    const realtype * x0,
    const realtype * p,
    const realtype * k,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsx0_fixedParameters

Parameters

<i>sx0</i>	initial state sensitivities
<i>t</i>	initial time
<i>x0</i>	initial state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>ip</i>	sensitivity index

Definition at line 1161 of file model.h.

10.11.3.133 fsx0() [2/2]

```
virtual void fsx0 (
    realtype * sx0,
    const realtype t,
    const realtype * x0,
    const realtype * p,
    const realtype * k,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsx0

Parameters

<i>sx0</i>	initial state sensitivities
<i>t</i>	initial time
<i>x0</i>	initial state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>ip</i>	sensitivity index

Definition at line 1172 of file model.h.

10.11.3.134 fstau() [2/2]

```
virtual void fstau (
    realtype * stau,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * sx,
    const int ip,
    const int ie ) [protected], [virtual]
```

model specific implementation of fstau

Parameters

<i>stau</i>	total derivative of event timepoint
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>sx</i>	current state sensitivity
<i>ip</i>	sensitivity index
<i>ie</i>	event index

Definition at line 1187 of file model.h.

10.11.3.135 fy() [2/2]

```
virtual void fy (
    realtype * y,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w ) [protected], [virtual]
```

model specific implementation of fy

Parameters

<i>y</i>	model output at current timepoint
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>w</i>	repeating elements vector

Definition at line 1200 of file model.h.

10.11.3.136 fdydp() [2/2]

```
virtual void fdydp (
    realtype * dydp,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const realtype * w,
    const realtype * dwdp ) [protected], [virtual]
```

model specific implementation of fdydp

Parameters

<i>dydp</i>	partial derivative of observables y w.r.t. model parameters p
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index w.r.t. which the derivative is requested
<i>w</i>	repeating elements vector

<i>Generated by</i>	Recovering terms in xdot, parameter derivative
---------------------	--

Definition at line 1215 of file model.h.

10.11.3.137 fdydx() [2/2]

```
virtual void fdydx (
    realtype * dydx,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation of fdydx

Parameters

<i>dydx</i>	partial derivative of observables y w.r.t. model states x
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>w</i>	repeating elements vector
<i>dwdx</i>	Recurring terms in xdot, state derivative

Definition at line 1229 of file model.h.

10.11.3.138 fz() [2/2]

```
virtual void fz (
    realtype * z,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation of fz

Parameters

<i>z</i>	value of event output
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector

Definition at line 1242 of file model.h.

10.11.3.139 fsz() [2/2]

```
virtual void fsz (
    realtype * sz,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * sx,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsz

Parameters

<i>sz</i>	Sensitivity of rz, total derivative
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>sx</i>	current state sensitivity
<i>ip</i>	sensitivity index

Definition at line 1257 of file model.h.

10.11.3.140 frz() [2/2]

```
virtual void frz (
    realtype * rz,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation of frz

Parameters

<i>rz</i>	value of root function at current timepoint (non-output events not included)
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector

Definition at line 1270 of file model.h.

10.11.3.141 fsrz() [2/2]

```
virtual void fsrz (
    realtype * srz,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * sx,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsrz

Parameters

<i>srz</i>	Sensitivity of rz, total derivative
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>sx</i>	current state sensitivity
<i>h</i>	heavyside vector
<i>ip</i>	sensitivity index

Definition at line 1285 of file model.h.

10.11.3.142 fdzdp() [2/2]

```
virtual void fdzdp (
    realtype * dzdp,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip ) [protected], [virtual]
```

model specific implementation of fdzdp

Parameters

<i>dzdp</i>	partial derivative of event-resolved output z w.r.t. model parameters p
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state

Parameters

<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index w.r.t. which the derivative is requested

Definition at line 1299 of file model.h.

10.11.3.143 fdzdx() [2/2]

```
virtual void fdzdx (
    realtype * dzdx,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation of fdzdx

Parameters

<i>dzdx</i>	partial derivative of event-resolved output z w.r.t. model states x
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector

Definition at line 1312 of file model.h.

10.11.3.144 fdrzdp() [2/2]

```
virtual void fdrzdp (
    realtype * drzdp,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip ) [protected], [virtual]
```

model specific implementation of fdrzdp

Parameters

<i>drzdp</i>	partial derivative of root output rz w.r.t. model parameters p
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index w.r.t. which the derivative is requested

Definition at line 1326 of file model.h.

10.11.3.145 fdrzdx() [2/2]

```
virtual void fdrzdx (
    realtype * drzdx,
    const int ie,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation of fdrzdx

Parameters

<i>drzdx</i>	partial derivative of root output rz w.r.t. model states x
<i>ie</i>	event index
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector

Definition at line 1339 of file model.h.

10.11.3.146 fdeltax() [2/2]

```
virtual void fdeltax (
    realtype * deltax,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ie,
```

```
const realtypes * xdot,  
const realtypes * xdot_old ) [protected], [virtual]
```

model specific implementation of fdeltax

Parameters

<i>deltax</i>	state update
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ie</i>	event index
<i>xdot</i>	new model right hand side
<i>xdot_old</i>	previous model right hand side

Definition at line 1354 of file model.h.

10.11.3.147 fdeltasx() [2/2]

```
virtual void fdeltasx (
    realtype * deltax,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w,
    const int ip,
    const int ie,
    const realtype * xdot,
    const realtype * xdot_old,
    const realtype * sx,
    const realtype * stau ) [protected], [virtual]
```

model specific implementation of fdeltasx

Parameters

<i>deltasx</i>	sensitivity update
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>w</i>	repeating elements vector
<i>ip</i>	sensitivity index
<i>ie</i>	event index
<i>xdot</i>	new model right hand side
<i>xdot_old</i>	previous model right hand side
<i>sx</i>	state sensitivity
<i>stau</i>	event-time sensitivity

Definition at line 1374 of file model.h.

10.11.3.148 fdeltaxB() [2/2]

```
virtual void fdeltaxB (
    realtype * deltaxB,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ie,
    const realtype * xdot,
    const realtype * xdot_old,
    const realtype * xB ) [protected], [virtual]
```

model specific implementation of fdeltaxB

Parameters

<i>deltaxB</i>	adjoint state update
<i>t</i>	current time
<i>x</i>	current state
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ie</i>	event index
<i>xdot</i>	new model right hand side
<i>xdot_old</i>	previous model right hand side
<i>xB</i>	current adjoint state

Definition at line 1392 of file model.h.

10.11.3.149 fdeltaqB() [2/2]

```
virtual void fdeltaqB (
    realtype * deltaqB,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const int ie,
    const realtype * xdot,
    const realtype * xdot_old,
    const realtype * xB ) [protected], [virtual]
```

model specific implementation of fdeltaqB

Parameters

<i>deltaqB</i>	sensitivity update
<i>t</i>	current time
<i>x</i>	current state

Parameters

<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>h</i>	heavyside vector
<i>ip</i>	sensitivity index
<i>ie</i>	event index
<i>xdot</i>	new model right hand side
<i>xdot_old</i>	previous model right hand side
<i>xB</i>	adjoint state

Definition at line 1410 of file model.h.

10.11.3.150 fsigmay() [2/2]

```
virtual void fsigmay (
    realtype * sigmay,
    const realtype t,
    const realtype * p,
    const realtype * k ) [protected], [virtual]
```

model specific implementation of fsigmay

Parameters

<i>sigmay</i>	standard deviation of measurements
<i>t</i>	current time
<i>p</i>	parameter vector
<i>k</i>	constant vector

Definition at line 1421 of file model.h.

10.11.3.151 fdsigmaydp() [2/2]

```
virtual void fdsigmaydp (
    realtype * dsigmaydp,
    const realtype t,
    const realtype * p,
    const realtype * k,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsigmay

Parameters

<i>dsigmaydp</i>	partial derivative of standard deviation of measurements
<i>t</i>	current time
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>ip</i>	sensitivity index

Definition at line 1432 of file model.h.

10.11.3.152 fsigmaz() [2/2]

```
virtual void fsigmaz (
    realtype * sigmaz,
    const realtype t,
    const realtype * p,
    const realtype * k ) [protected], [virtual]
```

model specific implementation of fsigmaz

Parameters

<i>sigmaz</i>	standard deviation of event measurements
<i>t</i>	current time
<i>p</i>	parameter vector
<i>k</i>	constant vector

Definition at line 1442 of file model.h.

10.11.3.153 fdsigmazdp() [2/2]

```
virtual void fdsigmazdp (
    realtype * dsigmazdp,
    const realtype t,
    const realtype * p,
    const realtype * k,
    const int ip ) [protected], [virtual]
```

model specific implementation of fsigmaz

Parameters

<i>dsigmazdp</i>	partial derivative of standard deviation of event measurements
<i>t</i>	current time
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>ip</i>	sensitivity index

Definition at line 1453 of file model.h.

10.11.3.154 fJy() [2/2]

```
virtual void fJy (
    realtype * nlh,
```

```

const int iy,
const realtype * p,
const realtype * k,
const realtype * y,
const realtype * sigmay,
const realtype * my ) [protected], [virtual]

```

model specific implementation of fJy

Parameters

<i>nllh</i>	negative log-likelihood for measurements y
<i>iy</i>	output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>y</i>	model output at timepoint
<i>sigmay</i>	measurement standard deviation at timepoint
<i>my</i>	measurements at timepoint

Definition at line 1466 of file model.h.

10.11.3.155 fJz() [2/2]

```

virtual void fJz (
    realtype * nllh,
    const int iz,
    const realtype * p,
    const realtype * k,
    const realtype * z,
    const realtype * sigmaz,
    const realtype * mz ) [protected], [virtual]

```

model specific implementation of fJz

Parameters

<i>nllh</i>	negative log-likelihood for event measurements z
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>z</i>	model event output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint
<i>mz</i>	event measurements at timepoint

Definition at line 1478 of file model.h.

10.11.3.156 fJrz() [2/2]

```

virtual void fJrz (
    realtype * nllh,

```

```

    const int iz,
    const realtype * p,
    const realtype * k,
    const realtype * z,
    const realtype * sigmaz ) [protected], [virtual]

```

model specific implementation of fJrz

Parameters

<i>nllh</i>	regularization for event measurements z
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>z</i>	model event output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint

Definition at line 1490 of file model.h.

10.11.3.157 fdJydy() [2/2]

```

virtual void fdJydy (
    realtype * dJydy,
    const int iy,
    const realtype * p,
    const realtype * k,
    const realtype * y,
    const realtype * sigmay,
    const realtype * my ) [protected], [virtual]

```

model specific implementation of fdJydy

Parameters

<i>dJydy</i>	partial derivative of time-resolved measurement negative log-likelihood Jy
<i>iy</i>	output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>y</i>	model output at timepoint
<i>sigmay</i>	measurement standard deviation at timepoint
<i>my</i>	measurement at timepoint

Definition at line 1503 of file model.h.

10.11.3.158 fdJydsigma() [2/2]

```

virtual void fdJydsigma (
    realtype * dJydsigma,
    const int iy,

```

```
const realtype * p,
const realtype * k,
const realtype * y,
const realtype * sigmay,
const realtype * my ) [protected], [virtual]
```

model specific implementation of fdJydsigma

Parameters

<i>dJydsigma</i>	Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigmay
<i>iy</i>	output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>y</i>	model output at timepoint
<i>sigmay</i>	measurement standard deviation at timepoint
<i>my</i>	measurement at timepoint

Definition at line 1518 of file model.h.

10.11.3.159 fdJzdz() [2/2]

```
virtual void fdJzdz (
    realtype * dJzdz,
    const int iz,
    const realtype * p,
    const realtype * k,
    const realtype * z,
    const realtype * sigmaz,
    const realtype * mz ) [protected], [virtual]
```

model specific implementation of fdJzdz

Parameters

<i>dJzdz</i>	partial derivative of event measurement negative log-likelihood Jz
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>z</i>	model event output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint
<i>mz</i>	event measurement at timepoint

Definition at line 1532 of file model.h.

10.11.3.160 fdJzdsigma() [2/2]

```
virtual void fdJzdsigma (
    realtype * dJzdsigma,
```

```

    const int iz,
    const realtype * p,
    const realtype * k,
    const realtype * z,
    const realtype * sigmaz,
    const realtype * mz ) [protected], [virtual]

```

model specific implementation of fdJzdsigma

Parameters

<i>dJzdsigma</i>	Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>z</i>	model event output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint
<i>mz</i>	event measurement at timepoint

Definition at line 1547 of file model.h.

10.11.3.161 fdJrzdz() [2/2]

```

virtual void fdJrzdz (
    realtype * dJrzdz,
    const int iz,
    const realtype * p,
    const realtype * k,
    const realtype * rz,
    const realtype * sigmaz ) [protected], [virtual]

```

model specific implementation of fdJrzdz

Parameters

<i>dJrzdz</i>	partial derivative of event penalization Jrz
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>rz</i>	model root output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint

Definition at line 1560 of file model.h.

10.11.3.162 fdJrzdsigma() [2/2]

```

virtual void fdJrzdsigma (
    realtype * dJrzdsigma,
    const int iz,

```

```
const realtype * p,
const realtype * k,
const realtype * rz,
const realtype * sigmaz ) [protected], [virtual]
```

model specific implementation of fdJrzdsigma

Parameters

<i>dJrzdsigma</i>	Sensitivity of event penalization Jrz w.r.t. standard deviation sigmaz
<i>iz</i>	event output index
<i>p</i>	parameter vector
<i>k</i>	constant vector
<i>rz</i>	model root output at timepoint
<i>sigmaz</i>	event measurement standard deviation at timepoint

Definition at line 1574 of file model.h.

10.11.3.163 fw() [2/2]

```
virtual void fw (
    realtype * w,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation of fw

Parameters

<i>w</i>	Recurring terms in xdot
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector

Definition at line 1587 of file model.h.

10.11.3.164 fdwdp() [2/2]

```
virtual void fdwdp (
    realtype * dwdp,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
```

```
const realtype * h,
const realtype * w) [protected], [virtual]
```

model specific implementation of dwdp

Parameters

<i>dwdp</i>	Recurring terms in xdot, parameter derivative
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables

Definition at line 1599 of file model.h.

10.11.3.165 fdwdx() [2/2]

```
virtual void fdwdx (
    realtype * dwdx,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w) [protected], [virtual]
```

model specific implementation of dwdx

Parameters

<i>dwdx</i>	Recurring terms in xdot, state derivative
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables

Definition at line 1611 of file model.h.

10.11.3.166 getmy()

```
void getmy (
    const int it,
    const ExpData * edata) [protected]
```

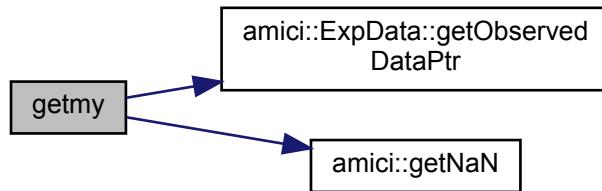
create my slice at timepoint

Parameters

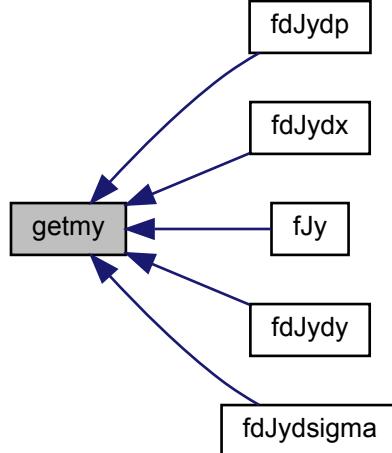
<i>it</i>	timepoint index
<i>edata</i>	pointer to experimental data instance

Definition at line 1229 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.167 getmz()**

```
void getmz (
    const int nroots,
    const ExpData * edata ) [protected]
```

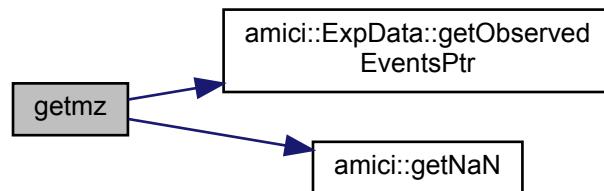
create mz slice at event

Parameters

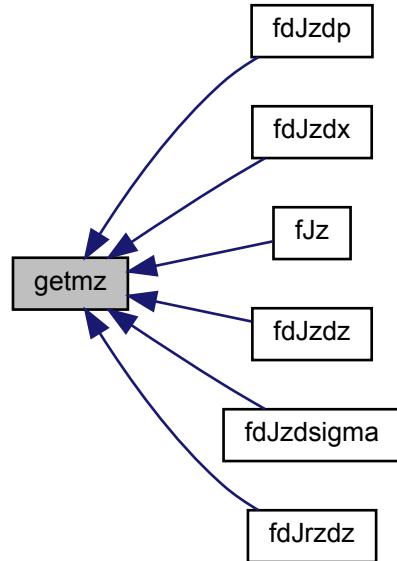
<i>nroots</i>	event occurrence
<i>edata</i>	pointer to experimental data instance

Definition at line 1253 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.3.168 gety()

```
const realtype * gety (
    const int it,
    const ReturnData * rdata ) const [protected]
```

create y slice at timepoint

Parameters

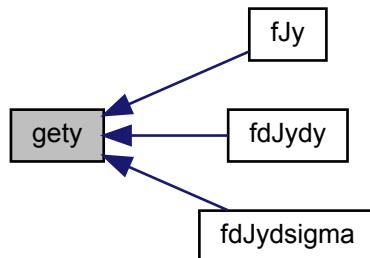
<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Returns

y y-slice from rdata instance

Definition at line 1245 of file model.cpp.

Here is the caller graph for this function:



10.11.3.169 getx()

```
const realtype * getx (
    const int it,
    const ReturnData * rdata ) const [protected]
```

create x slice at timepoint

Parameters

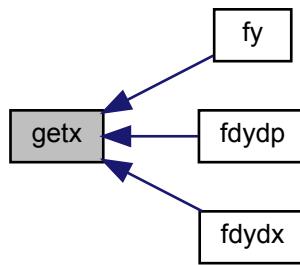
<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Returns

x x-slice from rdata instance

Definition at line 1237 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.170 getsx()**

```
const realtype * getsx (
    const int it,
    const ReturnData * rdata ) const [protected]
```

create sx slice at timepoint

Parameters

<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

Returns

sx sx-slice from rdata instance

Definition at line 1241 of file model.cpp.

Here is the call graph for this function:



10.11.3.171 getz()

```
const realtype * getz (
    const int nroots,
    const ReturnData * rdata ) const [protected]
```

create z slice at event

Parameters

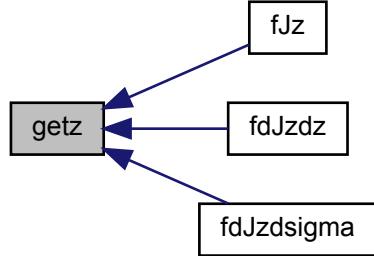
<i>nroots</i>	event occurrence
<i>rdata</i>	pointer to return data instance

Returns

z slice

Definition at line 1261 of file model.cpp.

Here is the caller graph for this function:



10.11.3.172 getrz()

```
const realtype * getrz (
    const int nroots,
    const ReturnData * rdata ) const [protected]
```

create rz slice at event

Parameters

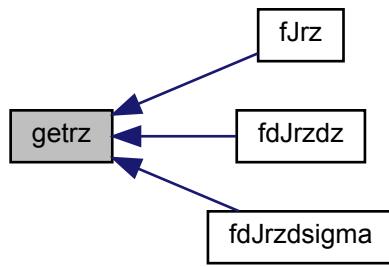
<i>nroots</i>	event occurrence
<i>rdata</i>	pointer to return data instance

Returns

rz slice

Definition at line 1265 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.173 getsz()**

```
const realtype * getsz (
    const int nroots,
    const int ip,
    const ReturnData * rdata ) const [protected]
```

create sz slice at event

Parameters

<code>nroots</code>	event occurrence
<code>ip</code>	sensitivity index
<code>rdata</code>	pointer to return data instance

Returns

`z slice`

Definition at line 1269 of file model.cpp.

Here is the call graph for this function:

**10.11.3.174 getsrz()**

```
const realtype * getsrz (
    const int nroots,
    const int ip,
    const ReturnData * rdata ) const [protected]
```

create srz slice at event

Parameters

<i>nroots</i>	event occurrence
<i>ip</i>	sensitivity index
<i>rdata</i>	pointer to return data instance

Returns

`rz slice`

Definition at line 1273 of file model.cpp.

Here is the call graph for this function:



10.11.3.175 isFixedParameterStateReinitializationAllowed()

```
virtual bool isFixedParameterStateReinitializationAllowed( ) const [protected], [virtual]
```

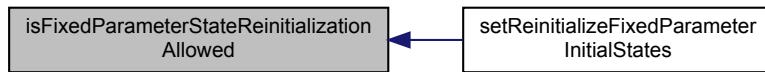
function indicating whether reinitialization of states depending on fixed parameters is permissible

Returns

flag indicating whether reinitialization of states depending on fixed parameters is permissible

Definition at line 1682 of file model.h.

Here is the caller graph for this function:

**10.11.3.176 computeX_pos()**

```
N_Vector computeX_pos ( N_Vector x ) [protected]
```

Parameters

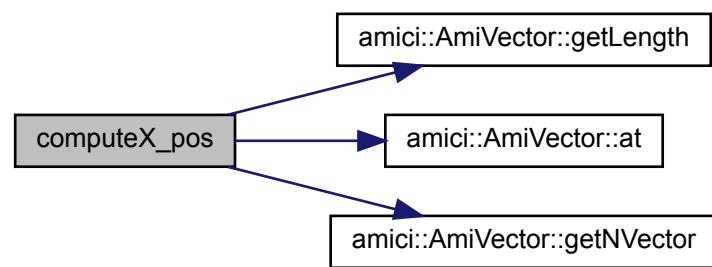
x	state vector possibly containing negative values
---	--

Returns

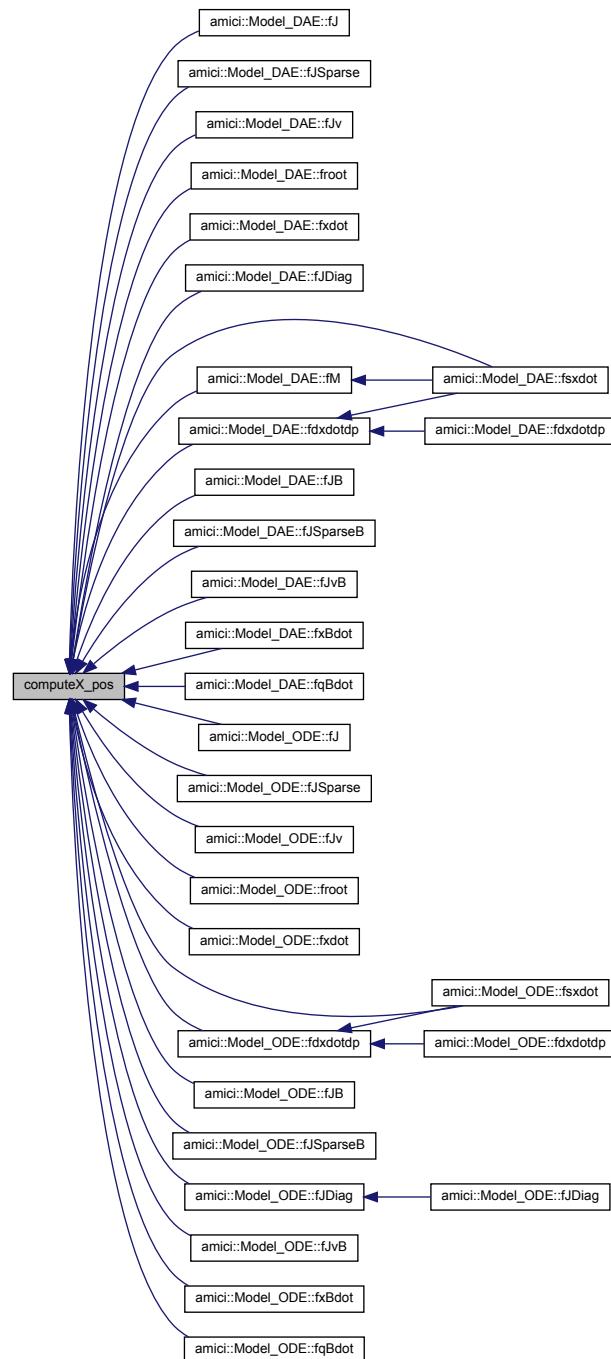
state vector with negative values replaced by 0 according to statelsNonNegative

Definition at line 1333 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.11.4 Friends And Related Function Documentation

10.11.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
    Archive & ar,
```

```
Model & r,
const unsigned int version ) [friend]
```

Parameters

<i>ar</i>	Archive to serialize to
<i>r</i>	Data to serialize
<i>version</i>	Version number

10.11.4.2 operator==

```
bool operator== (
    const Model & a,
    const Model & b ) [friend]
```

Parameters

<i>a</i>	first model instance
<i>b</i>	second model instance

Returns

equality

Definition at line 1297 of file model.cpp.

10.11.5 Member Data Documentation**10.11.5.1 nx**

const int nx

number of states

Definition at line 1065 of file model.h.

10.11.5.2 nxtrue

const int nxtrue

number of states in the unaugmented system

Definition at line 1067 of file model.h.

10.11.5.3 ny

```
const int ny
```

number of observables

Definition at line 1069 of file model.h.

10.11.5.4 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 1071 of file model.h.

10.11.5.5 nz

```
const int nz
```

number of event outputs

Definition at line 1073 of file model.h.

10.11.5.6 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 1075 of file model.h.

10.11.5.7 ne

```
const int ne
```

number of events

Definition at line 1077 of file model.h.

10.11.5.8 nw

const int nw

number of common expressions

Definition at line 1079 of file model.h.

10.11.5.9 ndwdx

const int ndwdx

number of derivatives of common expressions wrt x

Definition at line 1081 of file model.h.

10.11.5.10 ndwdp

const int ndwdp

number of derivatives of common expressions wrt p

Definition at line 1083 of file model.h.

10.11.5.11 nnz

const int nnz

number of nonzero entries in jacobian

Definition at line 1085 of file model.h.

10.11.5.12 nJ

const int nJ

dimension of the augmented objective function for 2nd order ASA

Definition at line 1087 of file model.h.

10.11.5.13 ubw

```
const int ubw
```

upper bandwith of the jacobian

Definition at line 1089 of file model.h.

10.11.5.14 lbw

```
const int lbw
```

lower bandwith of the jacobian

Definition at line 1091 of file model.h.

10.11.5.15 o2mode

```
const SecondOrderMode o2mode
```

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

Definition at line 1094 of file model.h.

10.11.5.16 z2event

```
const std::vector<int> z2event
```

index indicating to which event an event output belongs

Definition at line 1096 of file model.h.

10.11.5.17 idlist

```
const std::vector<realtypes> idlist
```

flag array for DAE equations

Definition at line 1098 of file model.h.

10.11.5.18 sigmay

```
std::vector<realtyp> sigmay
```

data standard deviation for current timepoint (dimension: ny)

Definition at line 1101 of file model.h.

10.11.5.19 dsigmaydp

```
std::vector<realtyp> dsigmaydp
```

parameter derivative of data standard deviation for current timepoint (dimension: nplist x ny, row-major)

Definition at line 1103 of file model.h.

10.11.5.20 sigmaz

```
std::vector<realtyp> sigmaz
```

event standard deviation for current timepoint (dimension: nz)

Definition at line 1105 of file model.h.

10.11.5.21 dsigmazdp

```
std::vector<realtyp> dsigmazdp
```

parameter derivative of event standard deviation for current timepoint (dimension: nplist x nz, row-major)

Definition at line 1107 of file model.h.

10.11.5.22 dJydp

```
std::vector<realtyp> dJydp
```

parameter derivative of data likelihood for current timepoint (dimension: nplist x nJ, row-major)

Definition at line 1109 of file model.h.

10.11.5.23 dJzdp

```
std::vector<realtyp> dJzdp
```

parameter derivative of event likelihood for current timepoint (dimension: nplist x nJ, row-major)

Definition at line 1111 of file model.h.

10.11.5.24 deltax

```
std::vector<realtyp> deltax
```

change in x at current timepoint (dimension: nx)

Definition at line 1114 of file model.h.

10.11.5.25 deltasx

```
std::vector<realtyp> deltasx
```

change in sx at current timepoint (dimension: nplist x nx, row-major)

Definition at line 1116 of file model.h.

10.11.5.26 deltaxB

```
std::vector<realtyp> deltaxB
```

change in xB at current timepoint (dimension: nJ x nxtrue, row-major)

Definition at line 1118 of file model.h.

10.11.5.27 deltaqB

```
std::vector<realtyp> deltaqB
```

change in qB at current timepoint (dimension: nJ x nplist, row-major)

Definition at line 1120 of file model.h.

10.11.5.28 dxdotdp

```
std::vector<realtyp> dxdotdp
```

temporary storage of dxdotdp data across functions (dimension: nplist x nx, row-major)

Definition at line 1123 of file model.h.

10.11.5.29 J

```
SlsMat J = nullptr [protected]
```

Sparse Jacobian (dimension: nnz)

Definition at line 1688 of file model.h.

10.11.5.30 my

```
std::vector<realtyp> my [protected]
```

current observable (dimension: nytrue)

Definition at line 1691 of file model.h.

10.11.5.31 mz

```
std::vector<realtyp> mz [protected]
```

current event measurement (dimension: nztrue)

Definition at line 1693 of file model.h.

10.11.5.32 dJydy

```
std::vector<realtyp> dJydy [protected]
```

observable derivative of data likelihood (dimension nj x nytrue x ny, ordering = ?)

Definition at line 1695 of file model.h.

10.11.5.33 dJydsigma

```
std::vector<realtype> dJydsigma [protected]
```

observable sigma derivative of data likelihood (dimension nJ x nytrue x ny, ordering = ?)

Definition at line 1697 of file model.h.

10.11.5.34 dJzdz

```
std::vector<realtype> dJzdz [protected]
```

event ouput derivative of event likelihood (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1700 of file model.h.

10.11.5.35 dJzdsigma

```
std::vector<realtype> dJzdsigma [protected]
```

event sigma derivative of event likelihood (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1702 of file model.h.

10.11.5.36 dJrzdz

```
std::vector<realtype> dJrzdz [protected]
```

event ouput derivative of event likelihood at final timepoint (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1704 of file model.h.

10.11.5.37 dJrzdsigma

```
std::vector<realtype> dJrzdsigma [protected]
```

event sigma derivative of event likelihood at final timepoint (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1706 of file model.h.

10.11.5.38 dzdx

```
std::vector<realtyp> dzdx [protected]
```

state derivative of event output (dimension: nz * nx, ordering = ?)

Definition at line 1708 of file model.h.

10.11.5.39 dzdp

```
std::vector<realtyp> dzdp [protected]
```

parameter derivative of event output (dimension: nz * nplist, ordering = ?)

Definition at line 1710 of file model.h.

10.11.5.40 drzdx

```
std::vector<realtyp> drzdx [protected]
```

state derivative of event timepoint (dimension: nz * nx, ordering = ?)

Definition at line 1712 of file model.h.

10.11.5.41 drzdp

```
std::vector<realtyp> drzdp [protected]
```

parameter derivative of event timepoint (dimension: nz * nplist, ordering = ?)

Definition at line 1714 of file model.h.

10.11.5.42 dydp

```
std::vector<realtyp> dydp [protected]
```

parameter derivative of observable (dimension: nplist * ny, row-major)

Definition at line 1716 of file model.h.

10.11.5.43 dydx

```
std::vector<realtyp> dydx [protected]
```

state derivative of observable (dimension: ny * nx, ordering = ?)

Definition at line 1719 of file model.h.

10.11.5.44 w

```
std::vector<realtyp> w [protected]
```

tempory storage of w data across functions (dimension: nw)

Definition at line 1721 of file model.h.

10.11.5.45 dwdx

```
std::vector<realtyp> dwdx [protected]
```

tempory storage of sparse dwdx data across functions (dimension: ndwdx)

Definition at line 1723 of file model.h.

10.11.5.46 dwdp

```
std::vector<realtyp> dwdp [protected]
```

tempory storage of sparse dwdp data across functions (dimension: ndwdp)

Definition at line 1725 of file model.h.

10.11.5.47 M

```
std::vector<realtyp> M [protected]
```

tempory storage of M data across functions (dimension: nx)

Definition at line 1727 of file model.h.

10.11.5.48 stau

```
std::vector<realtyp> stau [protected]
```

temporary storage of stau data across functions (dimension: nplist)

Definition at line 1729 of file model.h.

10.11.5.49 h

```
std::vector<realtyp> h [protected]
```

flag indicating whether a certain heaviside function should be active or not (dimension: ne)

Definition at line 1733 of file model.h.

10.11.5.50 unscaledParameters

```
std::vector<realtyp> unscaledParameters [protected]
```

unscaled parameters (dimension: np)

Definition at line 1736 of file model.h.

10.11.5.51 originalParameters

```
std::vector<realtyp> originalParameters [protected]
```

original user-provided, possibly scaled parameter array (size np)

Definition at line 1739 of file model.h.

10.11.5.52 fixedParameters

```
std::vector<realtyp> fixedParameters [protected]
```

constants (dimension: nk)

Definition at line 1742 of file model.h.

10.11.5.53 plist_

```
std::vector<int> plist_ [protected]
```

indexes of parameters wrt to which sensitivities are computed (dimension nplist)

Definition at line 1745 of file model.h.

10.11.5.54 x0data

```
std::vector<double> x0data [protected]
```

state initialisation (size nx)

Definition at line 1748 of file model.h.

10.11.5.55 sx0data

```
std::vector<realtyp> sx0data [protected]
```

sensitivity initialisation (size nx * nplist, ordering = ?)

Definition at line 1751 of file model.h.

10.11.5.56 ts

```
std::vector<realtyp> ts [protected]
```

timepoints (size nt)

Definition at line 1754 of file model.h.

10.11.5.57 stateIsNonNegative

```
std::vector<bool> stateIsNonNegative [protected]
```

vector of bools indicating whether state variables are to be assumed to be positive

Definition at line 1757 of file model.h.

10.11.5.58 anyStateNonNegative

```
bool anyStateNonNegative = false [protected]
```

boolean indicating whether any entry in statelsNonNegative is true

Definition at line 1760 of file model.h.

10.11.5.59 x_pos_tmp

```
AmiVector x_pos_tmp [protected]
```

temporary storage of positified state variables according to statelsNonNegative

Definition at line 1763 of file model.h.

10.11.5.60 nmaxevent

```
int nmaxevent = 10 [protected]
```

maximal number of events to track

Definition at line 1766 of file model.h.

10.11.5.61 pscale

```
std::vector<ParameterScaling> pscale [protected]
```

parameter transformation of originalParameters (dimension np)

Definition at line 1769 of file model.h.

10.11.5.62 tstart

```
double tstart = 0.0 [protected]
```

starting time

Definition at line 1772 of file model.h.

10.11.5.63 steadyStateSensitivityMode

```
SteadyStateSensitivityMode steadyStateSensitivityMode = SteadyStateSensitivityMode::newtonOnly  
[protected]
```

flag indicating whether steadystate sensitivities are to be computed via FSA when steadyStateSimulation is used

Definition at line 1776 of file model.h.

10.11.5.64 reinitializeFixedParameterInitialStates

```
bool reinitializeFixedParameterInitialStates = false [protected]
```

flag indicating whether reinitialization of states depending on fixed parameters is activated

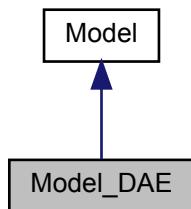
Definition at line 1781 of file model.h.

10.12 Model_DAE Class Reference

The [Model](#) class represents an AMICI DAE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.

```
#include <model_dae.h>
```

Inheritance diagram for Model_DAE:



Public Member Functions

- `Model_DAE ()`
- `Model_DAE (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 nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const SecondOrderMode o2mode, const std::vector< realtype > p, const std::vector< realtype > k, const std::vector< int > plist, const std::vector< realtype > idlist, const std::vector< int > z2event)`
- `virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J) override`
- `void fJ (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xdot, DlsMat J)`
- `void fJB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, DlsMat JB)`
- `virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J) override`
- `void fJSparse (realtype t, realtype cj, N_Vector x, N_Vector dx, SlsMat J)`
- `void fJSparseB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, SlsMat JB)`
- `virtual void fJDiag (realtype t, AmiVector *JDiag, realtype cj, AmiVector *x, AmiVector *dx) override`
- `virtual void fJv (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj) override`
- `void fJv (realtype t, N_Vector x, N_Vector dx, N_Vector v, N_Vector Jv, realtype cj)`
- `void fJvB (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector vB, N_Vector JvB, realtype cj)`
- `virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root) override`
- `void froot (realtype t, N_Vector x, N_Vector dx, realtype *root)`
- `virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot) override`
- `void fxdot (realtype t, N_Vector x, N_Vector dx, N_Vector xdot)`
- `void fxBDot (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector xBDot)`
- `void fqBDot (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector qBDot)`
- `void fxdotdp (const realtype t, const N_Vector x, const N_Vector dx)`
- `virtual void fxdotdp (realtype t, AmiVector *x, AmiVector *dx) override`
- `void fsxdot (realtype t, AmiVector *x, AmiVector *dx, int ip, AmiVector *sx, AmiVector *sdx, AmiVector *sxdot) override`
- `void fsxdot (realtype t, N_Vector x, N_Vector dx, int ip, N_Vector sx, N_Vector sdx, N_Vector sxdot)`
- `void fM (realtype t, const N_Vector x)`

Mass matrix for DAE systems.
- `virtual std::unique_ptr< Solver > getSolver () override`

Protected Member Functions

- `virtual void fJ (realtype *J, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)=0`
- `virtual void fJB (realtype *JB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxF, const realtype *w, const realtype *dwdx)`
- `virtual void fJSparse (SlsMat JSparse, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)=0`
- `virtual void fJSparseB (SlsMat JSparseB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxF, const realtype *w, const realtype *dwdx)`
- `virtual void fJDiag (realtype *JDiag, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)`
- `virtual void fJv (realtype *Jv, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *v, const realtype *w, const realtype *dwdx)`
- `virtual void fJvB (realtype *JvB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxF, const realtype *vB, const realtype *w, const realtype *dwdx)`
- `virtual void froot (realtype *root, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype *dx)`

- virtual void `fxdot` (`realtype` *`x`, const `realtype` `t`, const `realtype` *`x`, const double *`p`, const double *`k`, const `realtype` *`h`, const `realtype` *`dx`, const `realtype` *`w`)=0
- virtual void `fxBdot` (`realtype` *`x`, const `realtype` `t`, const `realtype` *`x`, const double *`p`, const double *`k`, const `realtype` *`h`, const `realtype` *`x`, const `realtype` *`dx`, const `realtype` *`dx`, const `realtype` *`w`, const `realtype` *`dwdx`)
- virtual void `fqBdot` (`realtype` *`q`, const int `ip`, const `realtype` `t`, const `realtype` *`x`, const double *`p`, const double *`k`, const `realtype` *`h`, const `realtype` *`x`, const `realtype` *`dx`, const `realtype` *`dx`, const `realtype` *`w`, const `realtype` *`dwdp`)
- virtual void `fdxdotdp` (`realtype` *`dx`, const `realtype` `t`, const `realtype` *`x`, const `realtype` *`p`, const `realtype` *`k`, const `realtype` *`h`, const int `ip`, const `realtype` *`dx`, const `realtype` *`w`, const `realtype` *`dwdp`)
- virtual void `fsxdot` (`realtype` *`s`, const `realtype` `t`, const `realtype` *`x`, const `realtype` *`p`, const `realtype` *`k`, const `realtype` *`h`, const int `ip`, const `realtype` *`dx`, const `realtype` *`s`, const `realtype` *`sd`, const `realtype` *`w`, const `realtype` *`dwdx`, const `realtype` *`M`, const `realtype` *`J`, const `realtype` *`dx`)
- virtual void `fM` (`realtype` *`M`, const `realtype` `t`, const `realtype` *`x`, const `realtype` *`p`, const `realtype` *`k`)

Additional Inherited Members

10.12.1 Detailed Description

Definition at line 24 of file model_dae.h.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 Model_DAE() [1/2]

`Model_DAE` ()

default constructor

Definition at line 27 of file model_dae.h.

10.12.2.2 Model_DAE() [2/2]

```
Model_DAE (
    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 nw,
    const int ndwdx,
    const int ndwdp,
    const int nnz,
    const int ubw,
    const int lbw,
    const SecondOrderMode o2mode,
    const std::vector< realtype > p,
    const std::vector< realtype > k,
    const std::vector< int > plist,
    const std::vector< realtype > idlist,
    const std::vector< int > z2event )
```

constructor with model dimensions

Parameters

<i>nx</i>	number of state variables
<i>nxtrue</i>	number of state variables of the non-augmented model
<i>ny</i>	number of observables
<i>nytrue</i>	number of observables of the non-augmented model
<i>nz</i>	number of event observables
<i>nztrue</i>	number of event observables of the non-augmented model
<i>ne</i>	number of events
<i>nJ</i>	number of objective functions
<i>nw</i>	number of repeating elements
<i>ndwdx</i>	number of nonzero elements in the x derivative of the repeating elements
<i>ndwdp</i>	number of nonzero elements in the p derivative of the repeating elements
<i>nnz</i>	number of nonzero elements in Jacobian
<i>ubw</i>	upper matrix bandwidth in the Jacobian
<i>lbw</i>	lower matrix bandwidth in the Jacobian
<i>o2mode</i>	second order sensitivity mode
<i>p</i>	parameters
<i>k</i>	constants
<i>plist</i>	indexes wrt to which sensitivities are to be computed
<i>idlist</i>	indexes indicating algebraic components (DAE only)
<i>z2event</i>	mapping of event outputs to events

Definition at line 53 of file `model_dae.h`.

10.12.3 Member Function Documentation

10.12.3.1 `fJ()` [1/3]

```
void fJ (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    DlsMat J ) [override], [virtual]
```

Dense Jacobian function

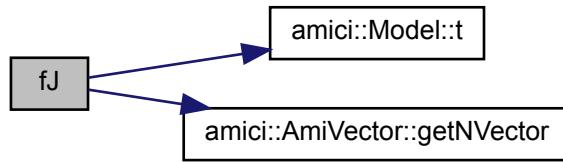
Parameters

<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	dense matrix to which values of the jacobian will be written

Implements [Model](#).

Definition at line 6 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.2 fJ() [2/3]

```
void fJ (
    realtype t,
    realtype cj,
    N_Vector x,
    N_Vector dx,
    N_Vector xdot,
    DlsMat J )
```

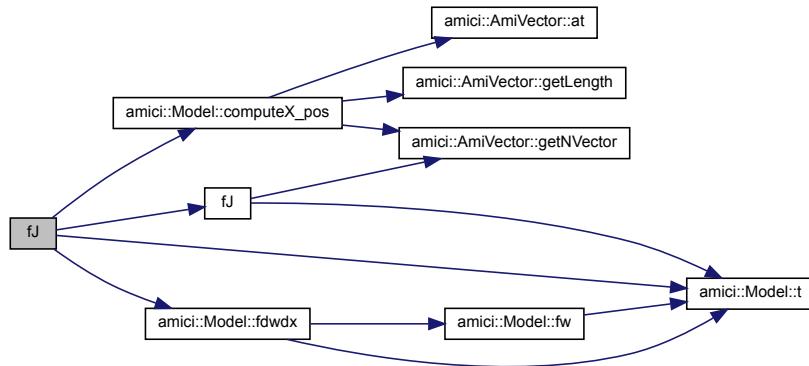
Jacobian of x_{dot} with respect to states x

Parameters

t	timepoint
cj	scaling factor, inverse of the step size
x	Vector with the states
dx	Vector with the derivative states
x_{dot}	Vector with the right hand side
J	Matrix to which the Jacobian will be written

Definition at line 20 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.3 fJB() [1/2]

```
void fJB (
    realtype t,
    realtype cj,
    N_Vector x,
    N_Vector dx,
    N_Vector xB,
    N_Vector dxB,
    DlsMat JB )
```

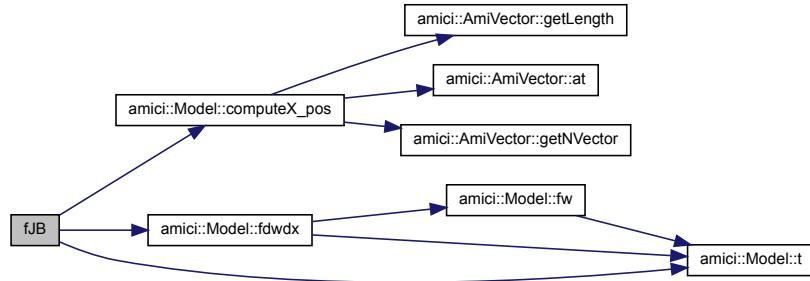
Jacobian of \dot{x}_B with respect to adjoint state x_B

Parameters

t	timepoint
cj	scaling factor, inverse of the step size
x	Vector with the states
dx	Vector with the derivative states
x_B	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
JB	Matrix to which the Jacobian will be written

Definition at line 166 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.4 fJSparse() [1/3]

```
void fJSparse (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    SlsMat J ) [override], [virtual]
```

Sparse Jacobian function

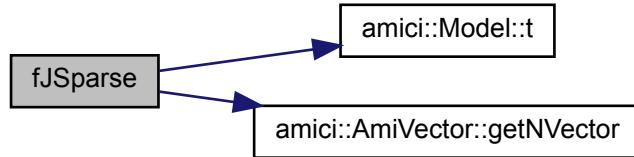
Parameters

<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	sparse matrix to which values of the Jacobian will be written

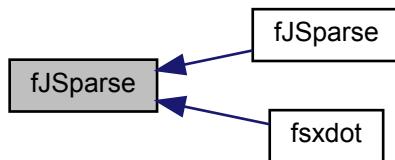
Implements [Model](#).

Definition at line 29 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.5 fJSparse() [2/3]

```
void fJSparse (
    realtype t,
    realtype cj,
    N_Vecor x,
    N_Vecor dx,
    SlsMat J )
```

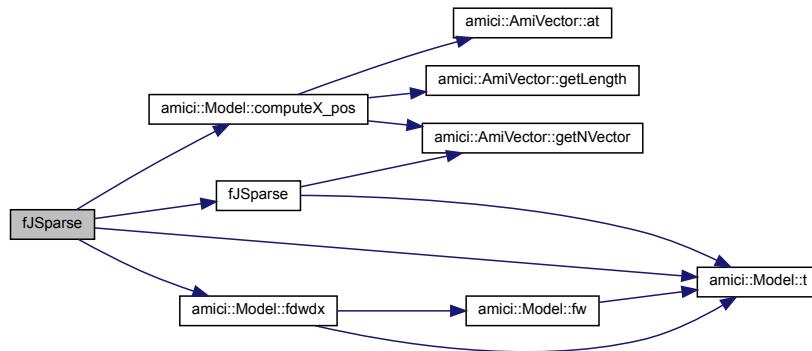
J in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

<i>t</i>	timepoint
<i>cj</i>	scalar in Jacobian (inverse stepsize)
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>J</i>	Matrix to which the Jacobian will be written

Definition at line 41 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.6 fJSparseB() [1/2]

```
void fJSparseB (
    realtype t,
    realtype cj,
    N_Vector x,
    N_Vector dx,
    N_Vector xB,
    N_Vector dxB,
    SlsMat JB )
```

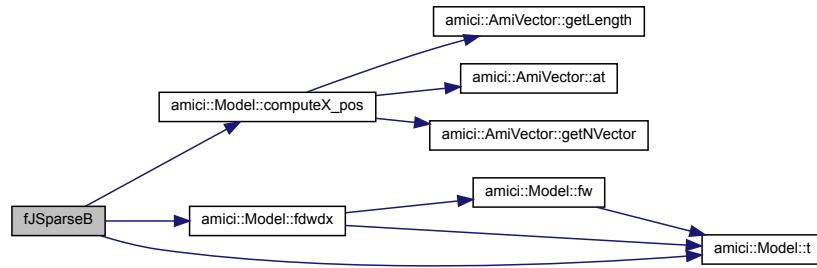
JB in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

<i>t</i>	timepoint
<i>cj</i>	scalar in Jacobian
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>xB</i>	Vector with the adjoint states
<i>dxB</i>	Vector with the adjoint derivative states
<i>JB</i>	Matrix to which the Jacobian will be written

Definition at line 184 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.7 fJDiag() [1/2]

```
void fJDiag (
    realtype t,
    AmiVector * JDdiag,
    realtype cj,
    AmiVector * x,
    AmiVector * dx ) [override], [virtual]
```

diagonalized Jacobian (for preconditioning)

Parameters

<i>t</i>	timepoint
<i>JDdiag</i>	Vector to which the Jacobian diagonal will be written
<i>cj</i>	scaling factor, inverse of the step size
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states

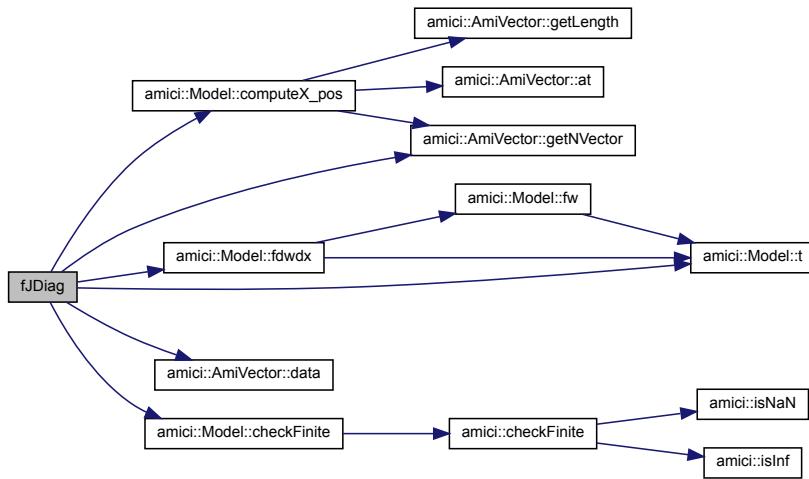
Returns

status flag indicating successful execution

Implements [Model](#).

Definition at line 116 of file `model_dae.cpp`.

Here is the call graph for this function:



10.12.3.8 fJv() [1/3]

```
void fJv (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    AmiVector * v,
    AmiVector * nJv,
    realtype cj ) [override], [virtual]
```

Jacobian multiply function

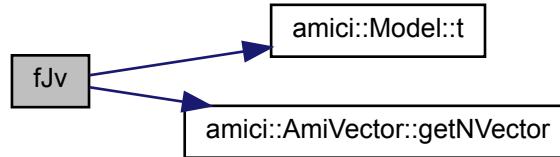
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>v</i>	multiplication vector (unused)
<i>nJv</i>	array to which result of multiplication will be written
<i>cj</i>	scaling factor (inverse of timestep, DAE only)

Implements [Model](#).

Definition at line 49 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.9 fJv() [2/3]

```
void fJv (
    realtype t,
    N_Vector x,
    N_Vector dx,
    N_Vector v,
    N_Vector Jv,
    realtype cj )
```

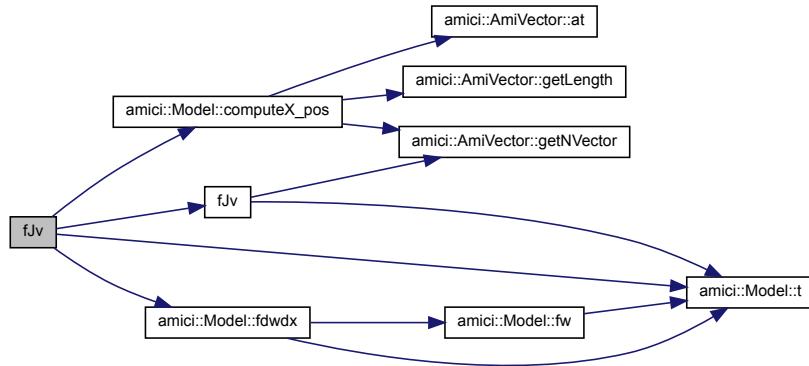
Matrix vector product of J with a vector v (for iterative solvers)

Parameters

<i>t</i>	timepoint Type: realtype
<i>cj</i>	scaling factor, inverse of the step size
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>v</i>	Vector with which the Jacobian is multiplied
<i>Jv</i>	Vector to which the Jacobian vector product will be written

Definition at line 64 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.10 fJvB() [1/2]

```
void fJvB (
    realtype t,
    N_Vector x,
    N_Vector dx,
    N_Vector xB,
    N_Vector dxB,
    N_Vector vB,
    N_Vector JvB,
    realtype cj )
```

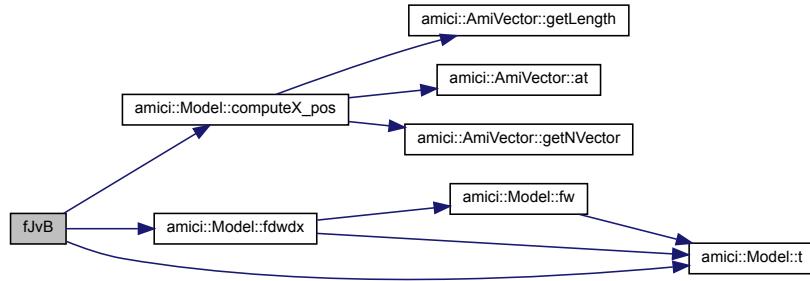
Matrix vector product of JB with a vector v (for iterative solvers)

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>xB</i>	Vector with the adjoint states
<i>dxB</i>	Vector with the adjoint derivative states
<i>vB</i>	Vector with which the Jacobian is multiplied
<i>JvB</i>	Vector to which the Jacobian vector product will be written
<i>cj</i>	scalar in Jacobian (inverse stepsize)

Definition at line 204 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.11 froot() [1/3]

```
void froot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    realtype * root )  [override], [virtual]
```

Root function

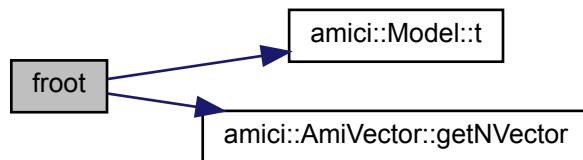
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>root</i>	array to which values of the root function will be written

Implements [Model](#).

Definition at line 73 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.12 froot() [2/3]

```
void froot (
    realtype t,
    N_Vector x,
    N_Vector dx,
    realtype * root )
```

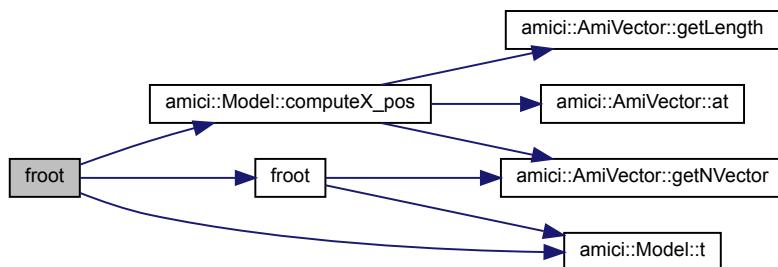
Event trigger function for events

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>root</i>	array with root function values

Definition at line 83 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.13 fxdot() [1/3]

```
void fxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot ) [override], [virtual]
```

Residual function

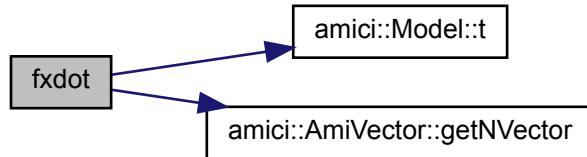
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	array to which values of the residual function will be written

Implements [Model](#).

Definition at line 90 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.14 fxdot() [2/3]

```
void fxdot (
    realtype t,
    N_Vector x,
    N_Vector dx,
    N_Vector xdot )
```

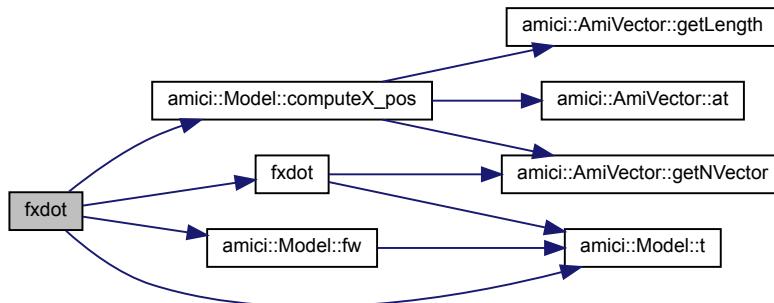
residual function of the DAE

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>xdot</i>	Vector with the right hand side

Definition at line 100 of file model_dae.cpp.

Here is the call graph for this function:

**10.12.3.15 fxBdot() [1/2]**

```
void fxBdot (
    realtype t,
    N_Vector x,
    N_Vector dx,
    N_Vector xB,
    N_Vector dxB,
    N_Vector xBdot )
```

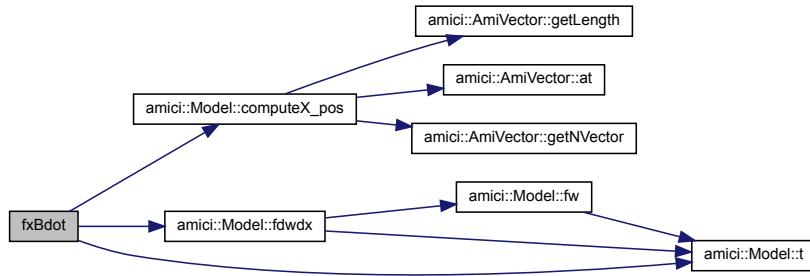
Right hand side of differential equation for adjoint state x_B

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>xB</i>	Vector with the adjoint states
<i>dxB</i>	Vector with the adjoint derivative states
<i>xBdot</i>	Vector with the adjoint right hand side

Definition at line 222 of file model_dae.cpp.

Here is the call graph for this function:



10.12.3.16 fqBdot() [1/2]

```
void fqBdot (
    realtype t,
    N_Vecor x,
    N_Vecor dx,
    N_Vecor xB,
    N_Vecor dxB,
    N_Vecor qBdot )
```

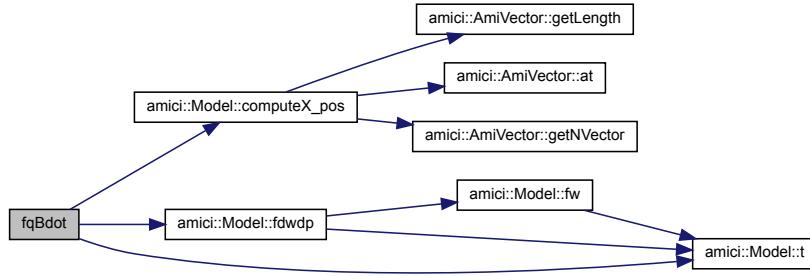
Right hand side of integral equation for quadrature states qB

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
$qBdot$	Vector with the adjoint quadrature right hand side

Definition at line 240 of file `model_dae.cpp`.

Here is the call graph for this function:



10.12.3.17 fdxdotdp() [1/3]

```
void fdxdotdp (
    const realltype t,
    const N_Vecor x,
    const N_Vecor dx )
```

Sensitivity of dx/dt wrt model parameters p

Parameters

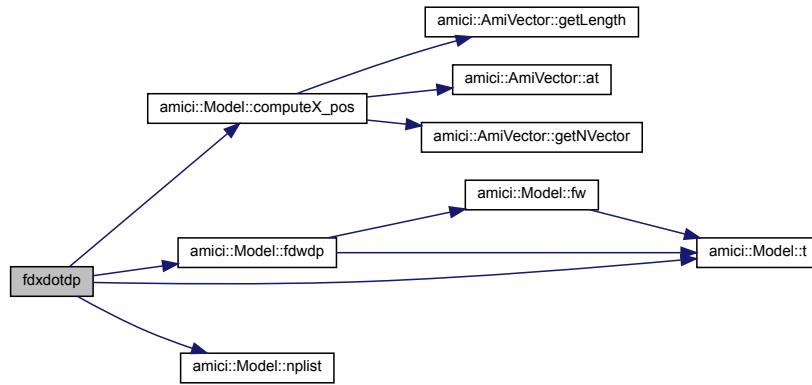
t	timepoint
x	Vector with the states
dx	Vector with the derivative states

Returns

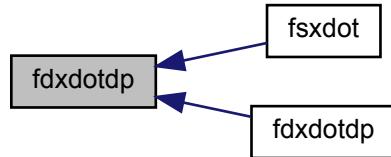
status flag indicating successful execution

Definition at line 133 of file model_dae.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.18 fdxdotdp() [2/3]

```

virtual void fdxdotdp (
    realscalar t,
    AmiVector * x,
    AmiVector * dx ) [override], [virtual]
  
```

parameter derivative of residual function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)

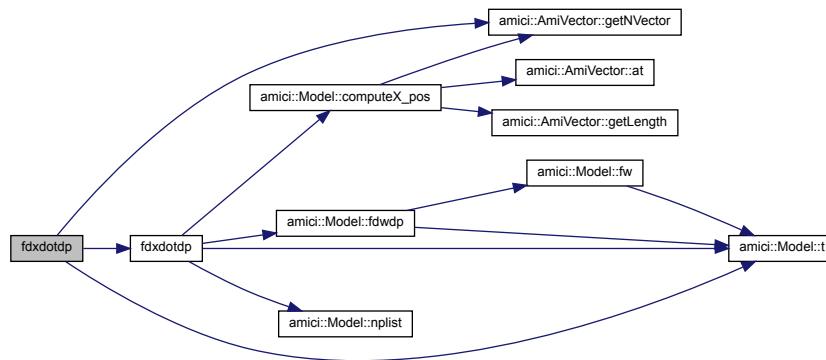
Returns

flag indicating successful evaluation

Implements [Model](#).

Definition at line 95 of file `model_dae.h`.

Here is the call graph for this function:

**10.12.3.19 fsxdot() [1/3]**

```
void fsxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    int ip,
    AmiVector * sx,
    AmiVector * sdx,
    AmiVector * sxdot )  [override], [virtual]
```

Sensitivity Residual function

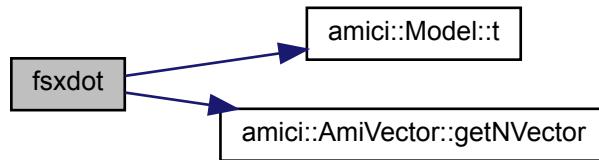
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>ip</i>	parameter index
<i>sx</i>	sensitivity state
<i>sdx</i>	time derivative of sensitivity state (DAE only)
<i>sxdot</i>	array to which values of the sensitivity residual function will be written

Implements [Model](#).

Definition at line 250 of file `model_dae.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.20 fsxdot() [2/3]

```

void fsxdot (
    realtype t,
    N_Vector x,
    N_Vector dx,
    int ip,
    N_Vector sx,
    N_Vector sdx,
    N_Vector sxdot )

```

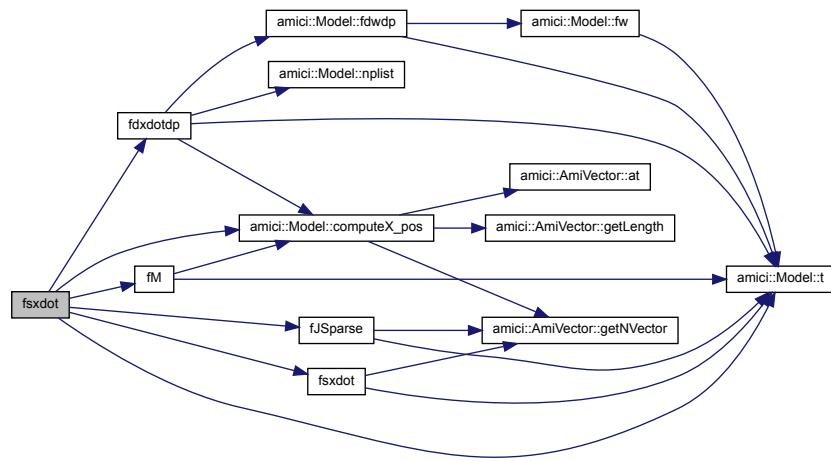
Right hand side of differential equation for state sensitivities sx

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states
<i>ip</i>	parameter index
<i>sx</i>	Vector with the state sensitivities
<i>sdx</i>	Vector with the derivative state sensitivities
<i>sxdot</i>	Vector with the sensitivity right hand side

Definition at line 266 of file model_dae.cpp.

Here is the call graph for this function:



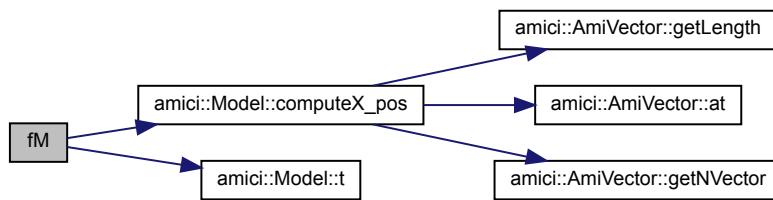
10.12.3.21 fM() [1/2]

```
void fM ( realtype t,  
          const N_Vector x )
```

Parameters

Definition at line 147 of file model_dae.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.12.3.22 getSolver()

```
std::unique_ptr< Solver > getSolver() [override], [virtual]
```

Retrieves the solver object

Returns

The [Solver](#) instance

Implements [Model](#).

Definition at line 153 of file `model_dae.cpp`.

10.12.3.23 fJ() [3/3]

```
virtual void fJ(
    realtype * J,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * dx,
    const realtype * w,
    const realtype * dwdx) [protected], [pure virtual]
```

model specific implementation for fJ

Parameters

<i>J</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>dx</i> Generated by Doxygen	Vector with the derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of <i>w</i> wrt <i>x</i>

10.12.3.24 fJB() [2/2]

```
virtual void fJB (
    realtype * JB,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * xB,
    const realtype * dx,
    const realtype * dxB,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJB

Parameters

<i>JB</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>xB</i>	Vector with the adjoint states
<i>dx</i>	Vector with the derivative states
<i>dxB</i>	Vector with the adjoint derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 139 of file model_dae.h.

10.12.3.25 fJSparse() [3/3]

```
virtual void fJSparse (
    SlsMat JSparse,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * dx,
    const realtype * w,
    const realtype * dwdx ) [protected], [pure virtual]
```

model specific implementation for fJSparse

Parameters

<i>JSparse</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>dx</i>	Vector with the derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

10.12.3.26 fJSparseB() [2/2]

```
virtual void fJSparseB (
    SlsMat JSparseB,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * xB,
    const realtype * dx,
    const realtype * dxB,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJSparseB

Parameters

<i>JSparseB</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>xB</i>	Vector with the adjoint states
<i>dx</i>	Vector with the derivative states
<i>dxB</i>	Vector with the adjoint derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 174 of file model_dae.h.

10.12.3.27 fJDiag() [2/2]

```
virtual void fJDiag (
    realtype * JDiag,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype cj,
    const realtype * dx,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJDiag

Parameters

<i>JDiag</i>	array to which the Jacobian diagonal will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>dx</i>	Vector with the derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 192 of file model_dae.h.

10.12.3.28 fJv() [3/3]

```
virtual void fJv (
    realtype * Jv,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * dx,
    const realtype * v,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJv

Parameters

<i>Jv</i>	Matrix vector product of J with a vector v
<i>t</i>	timepoint
<i>x</i>	Vector with the states

Parameters

<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>dx</i>	Vector with the derivative states
<i>v</i>	Vector with which the Jacobian is multiplied
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 210 of file model_dae.h.

10.12.3.29 fJvB() [2/2]

```
virtual void fJvB (
    realtype * JvB,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype cj,
    const realtype * xB,
    const realtype * dx,
    const realtype * dxB,
    const realtype * vB,
    const realtype * w,
    const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJvB

Parameters

<i>JvB</i>	Matrix vector product of JB with a vector v
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>cj</i>	scaling factor, inverse of the step size
<i>xB</i>	Vector with the adjoint states
<i>dx</i>	Vector with the derivative states
<i>dxB</i>	Vector with the adjoint derivative states
<i>vB</i>	Vector with which the Jacobian is multiplied
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 231 of file model_dae.h.

10.12.3.30 froot() [3/3]

```
virtual void froot (
    realtype * root,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype * dx ) [protected], [virtual]
```

model specific implementation for froot

Parameters

<i>root</i>	values of the trigger function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>dx</i>	Vector with the derivative states

Definition at line 246 of file model_dae.h.

10.12.3.31 fxdot() [3/3]

```
virtual void fxdot (
    realtype * xdot,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype * dx,
    const realtype * w ) [protected], [pure virtual]
```

model specific implementation for fxdot

Parameters

<i>xdot</i>	residual function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables
<i>dx</i>	Vector with the derivative states

10.12.3.32 fxBdot() [2/2]

```
virtual void fxBdot (
    realtype * xBdot,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype * xB,
    const realtype * dx,
    const realtype * dxB,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fxBdot

Parameters

<i>xBdot</i>	adjoint residual function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>dx</i>	Vector with the derivative states
<i>dxB</i>	Vector with the adjoint derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 277 of file model_dae.h.

10.12.3.33 fqBdot() [2/2]

```
virtual void fqBdot (
    realtype * qBdot,
    const int ip,
    const realtype t,
    const realtype * x,
    const double * p,
    const double * k,
    const realtype * h,
    const realtype * xB,
    const realtype * dx,
    const realtype * dxB,
    const realtype * w,
    const realtype * dwdp ) [protected], [virtual]
```

model specific implementation for fqBdot

Parameters

<i>qBdot</i>	adjoint quadrature equation
<i>ip</i>	sensitivity index
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>dx</i>	Vector with the derivative states
<i>dxB</i>	Vector with the adjoint derivative states
<i>w</i>	vector with helper variables
<i>dwdp</i>	derivative of w wrt p

Definition at line 297 of file model_dae.h.

10.12.3.34 fxdotdp() [3/3]

```
virtual void fxdotdp (
    realtype * dxdotdp,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const realtype * dx,
    const realtype * w,
    const realtype * dwdp ) [protected], [virtual]
```

model specific implementation of fxdotdp

Parameters

<i>dxdotdp</i>	partial derivative xdot wrt p
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index
<i>dx</i>	Vector with the derivative states
<i>w</i>	vector with helper variables
<i>dwdp</i>	derivative of w wrt p

Definition at line 315 of file model_dae.h.

10.12.3.35 fsxdot() [3/3]

```
virtual void fsxdot (
    realtype * sxdot,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const realtype * dx,
    const realtype * sx,
    const realtype * sdx,
    const realtype * w,
    const realtype * dwdx,
    const realtype * M,
    const realtype * J,
    const realtype * dxdotdp ) [protected], [virtual]
```

model specific implementation of fsxdot

Parameters

<i>sxdot</i>	sensitivity rhs
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index
<i>dx</i>	Vector with the derivative states
<i>sx</i>	Vector with the state sensitivities
<i>sdx</i>	Vector with the derivative state sensitivities
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x
<i>M</i>	mass matrix
<i>J</i>	jacobian
<i>dxdotdp</i>	parameter derivative of residual function

Definition at line 337 of file model_dae.h.

10.12.3.36 fM() [2/2]

```
virtual void fM (
    realtype * M,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k ) [protected], [virtual]
```

model specific implementation of fM

Parameters

M	mass matrix
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector

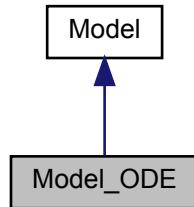
Definition at line 351 of file model_dae.h.

10.13 Model_ODE Class Reference

The [Model](#) class represents an AMICI ODE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.

```
#include <model_ode.h>
```

Inheritance diagram for Model_ODE:



Public Member Functions

- [`Model_ODE \(\)`](#)
- [`Model_ODE \(const int nx, const int nxtrue, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nj, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const SecondOrderMode o2mode, const std::vector< realtype > p, const std::vector< realtype > k, const std::vector< int > plist, const std::vector< realtype > idlist, const std::vector< int > z2event\)`](#)
- [`virtual void fJ \(realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J\) override`](#)
- [`void fJ \(realtype t, N_Vector x, N_Vector xdot, DlsMat J\)`](#)
- [`void fJB \(realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, DlsMat JB\)`](#)
- [`virtual void fJSparse \(realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J\) override`](#)
- [`void fJSparse \(realtype t, N_Vector x, SlsMat J\)`](#)
- [`void fJSparseB \(realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, SlsMat JB\)`](#)
- [`void fJDiag \(realtype t, N_Vector JDdiag, N_Vector x\)`](#)
- [`virtual void fJDiag \(realtype t, AmiVector *JDdiag, realtype cj, AmiVector *x, AmiVector *dx\) override`](#)
- [`virtual void fJv \(realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj\) override`](#)
- [`void fJv \(N_Vector v, N_Vector Jv, realtype t, N_Vector x\)`](#)
- [`void fJvB \(N_Vector vB, N_Vector JvB, realtype t, N_Vector x, N_Vector xB\)`](#)

- virtual void `froot` (realtype t, AmiVector *x, AmiVector *dx, realtype *root) override
- void `froot` (realtype t, N_Vector x, realtype *root)
- virtual void `fxdot` (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot) override
- void `fxdot` (realtype t, N_Vector x, N_Vector xdot)
- void `fxBdot` (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot)
- void `fqBdot` (realtype t, N_Vector x, N_Vector xB, N_Vector qBdot)
- void `fdxdotdp` (const realtype t, const N_Vector x)
- virtual void `fdxdotdp` (realtype t, AmiVector *x, AmiVector *dx) override
- void `fsxdot` (realtype t, AmiVector *x, AmiVector *dx, int ip, AmiVector *sx, AmiVector *sdx, AmiVector *sxdot) override
- void `fsxdot` (realtype t, N_Vector x, int ip, N_Vector sx, N_Vector sxdot)
- virtual std::unique_ptr< Solver > `getSolver` () override

Protected Member Functions

- virtual void `fJ` (realtype *J, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)=0
- virtual void `fJB` (realtype *JB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void `fJSparse` (SlsMat JSparse, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)=0
- virtual void `fJSparseB` (SlsMat JSparseB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void `fJDiag` (realtype *JDiag, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)
- virtual void `fJv` (realtype *Jv, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *v, const realtype *w, const realtype *dwdx)
- virtual void `fJvB` (realtype *JvB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *vB, const realtype *w, const realtype *dwdx)
- virtual void `froot` (realtype *root, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void `fxdot` (realtype *xdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)=0
- virtual void `fxBdot` (realtype *xBdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void `fqBdot` (realtype *qBdot, const int ip, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdp)
- virtual void `fdxdotdp` (realtype *dxdotdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *w, const realtype *dwdp)
- virtual void `fsxdot` (realtype *sxdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *sx, const realtype *w, const realtype *dwdx, const realtype *J, const realtype *dxdotdp)

Additional Inherited Members

10.13.1 Detailed Description

Definition at line 23 of file model_ode.h.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 Model_ODE() [1/2]

`Model_ODE ()`

default constructor

Definition at line 26 of file `model_ode.h`.

10.13.2.2 Model_ODE() [2/2]

```
Model_ODE (
    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 nw,
    const int ndwdx,
    const int ndwdp,
    const int nnz,
    const int ubw,
    const int lbw,
    const SecondOrderMode o2mode,
    const std::vector< realtype > p,
    const std::vector< realtype > k,
    const std::vector< int > plist,
    const std::vector< realtype > idlist,
    const std::vector< int > z2event )
```

constructor with model dimensions

Parameters

<code>nx</code>	number of state variables
<code>nxtrue</code>	number of state variables of the non-augmented model
<code>ny</code>	number of observables
<code>nytrue</code>	number of observables of the non-augmented model
<code>nz</code>	number of event observables
<code>nztrue</code>	number of event observables of the non-augmented model
<code>ne</code>	number of events
<code>nJ</code>	number of objective functions
<code>nw</code>	number of repeating elements
<code>ndwdx</code>	number of nonzero elements in the x derivative of the repeating elements
<code>ndwdp</code>	number of nonzero elements in the p derivative of the repeating elements
<code>nnz</code>	number of nonzero elements in Jacobian
<code>ubw</code>	upper matrix bandwidth in the Jacobian
<code>lbw</code>	lower matrix bandwidth in the Jacobian
<code>o2mode</code>	second order sensitivity mode
<code>p</code>	parameters
<code>k</code>	constants
<code>plist</code>	indexes wrt to which sensitivities are to be computed
<code>idlist</code>	indexes indicating algebraic components (DAE only)
<code>z2event</code>	mapping of event outputs to events

Definition at line 52 of file model_ode.h.

10.13.3 Member Function Documentation

10.13.3.1 fJ() [1/3]

```
void fJ (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    DlsMat J ) [override], [virtual]
```

Dense Jacobian function

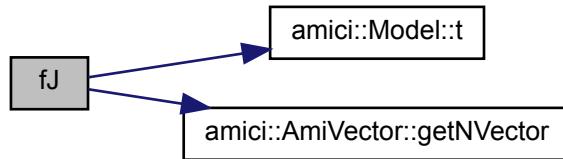
Parameters

<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	dense matrix to which values of the jacobian will be written

Implements [Model](#).

Definition at line 6 of file model_ode.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.2 fJ() [2/3]

```
void fJ (
    realtype t,
    N_Vector x,
    N_Vector xdot,
    DlsMat J )
```

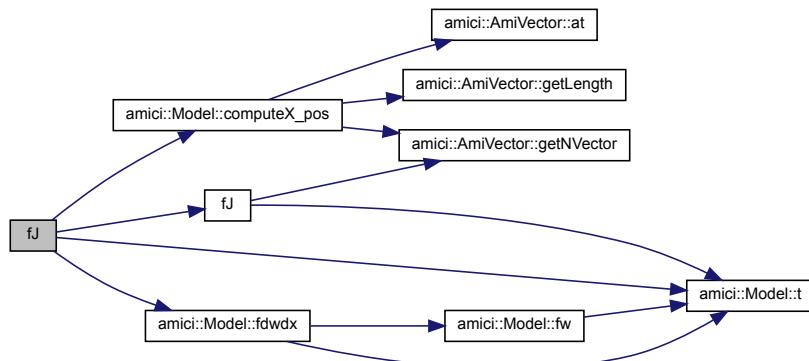
implementation of fJ at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xdot</i>	Vector with the right hand side
<i>J</i>	Matrix to which the Jacobian will be written

Definition at line 20 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.3 fJB() [1/2]

```
void fJB (
    realtype t,
    N_Vector x,
    N_Vector xB,
    N_Vector xBdot,
    DlsMat JB )
```

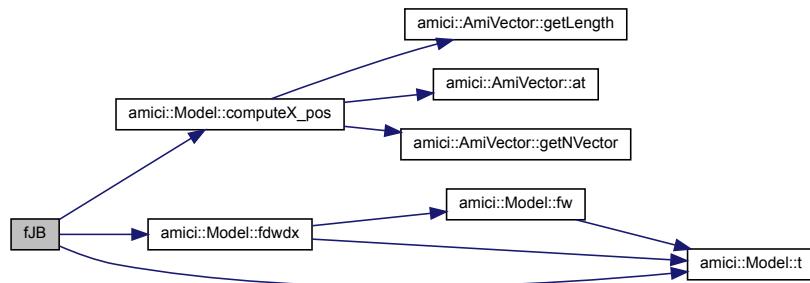
implementation of fJB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xB</i>	Vector with the adjoint states
<i>xBdot</i>	Vector with the adjoint right hand side
<i>JB</i>	Matrix to which the Jacobian will be written

Definition at line 147 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.4 fJSparse() [1/3]

```
void fJSparse (
    realtype t,
    realtype cj,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    SlsMat J ) [override], [virtual]
```

Sparse Jacobian function

Parameters

<i>t</i>	time
----------	------

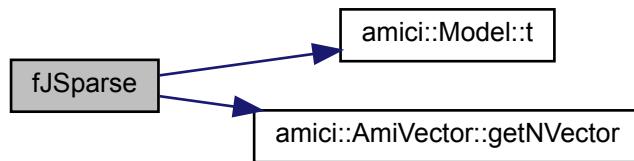
Parameters

<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>J</i>	sparse matrix to which values of the Jacobian will be written

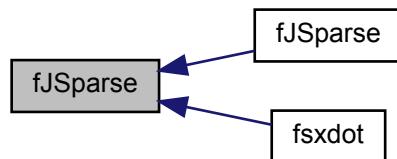
Implements [Model](#).

Definition at line 28 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.5 `fJSparse()` [2/3]

```

void fJSparse (
    realtype t,
    N_Vecor x,
    SlsMat J )
  
```

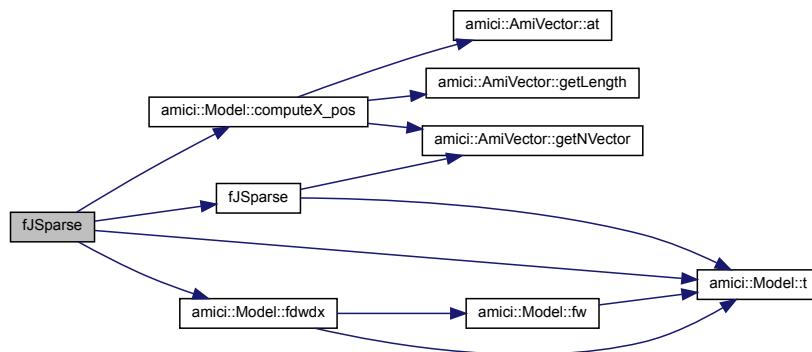
implementation of `fJSparse` at the `N_Vecor` level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>J</i>	Matrix to which the Jacobian will be written

Definition at line 40 of file model_ode.cpp.

Here is the call graph for this function:

**10.13.3.6 fJSparseB() [1/2]**

```
void fJSparseB (
    realtype t,
    N_Vecor x,
    N_Vecor xB,
    N_Vecor xBdot,
    SlsMat JB )
```

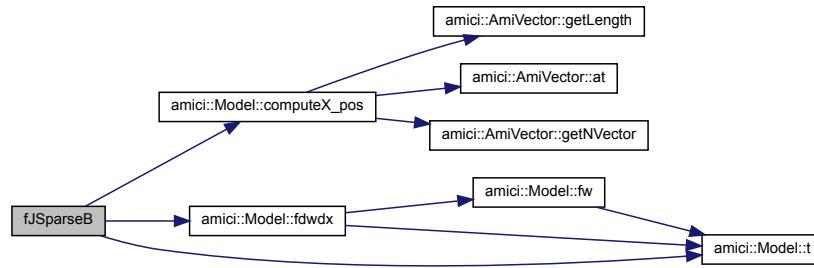
implementation of fJSparseB at the N_Vecor level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xB</i>	Vector with the adjoint states
<i>xBdot</i>	Vector with the adjoint right hand side
<i>JB</i>	Matrix to which the Jacobian will be written

Definition at line 164 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.7 fJDiag() [1/3]

```
void fJDiag (
    realtype t,
    N_Vecor JDiag,
    N_Vecor x )
```

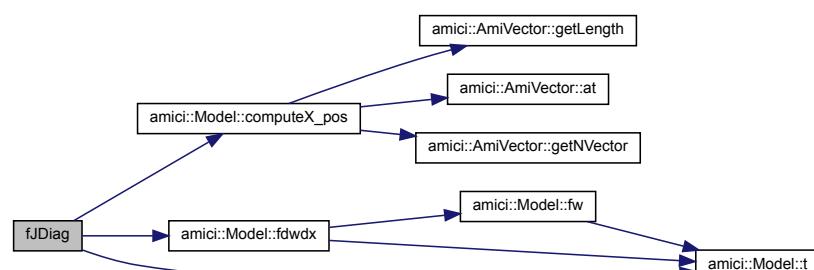
implementation of fJDiag at the N_Vecor level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>JDiag</i>	Vector to which the Jacobian diagonal will be written
<i>x</i>	Vector with the states

Definition at line 178 of file model_ode.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.8 fJDiag() [2/3]

```
void fJDiag (
    realtype t,
    AmiVector * JDdiag,
    realtype cj,
    AmiVector * x,
    AmiVector * dx ) [override], [virtual]
```

diagonalized Jacobian (for preconditioning)

Parameters

<i>t</i>	timepoint
<i>JDdiag</i>	Vector to which the Jacobian diagonal will be written
<i>cj</i>	scaling factor, inverse of the step size
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states

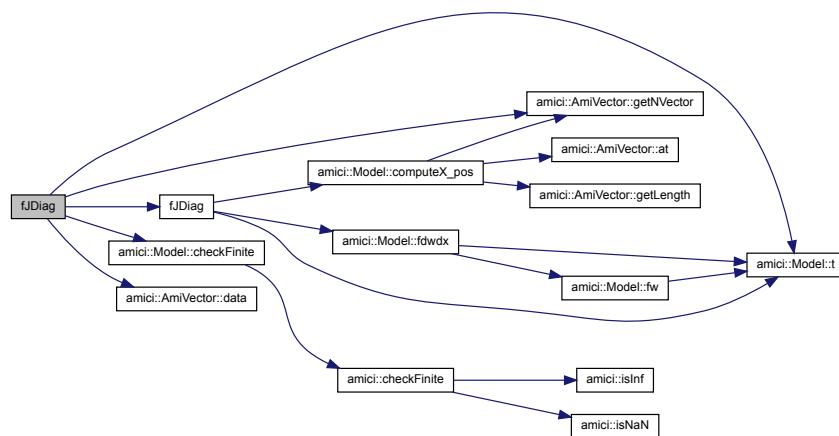
Returns

status flag indicating successful execution

Implements [Model](#).

Definition at line 114 of file `model_ode.cpp`.

Here is the call graph for this function:



10.13.3.9 fJv() [1/3]

```
void fJv (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot,
    AmiVector * v,
    AmiVector * nJv,
    realtype cj ) [override], [virtual]
```

Jacobian multiply function

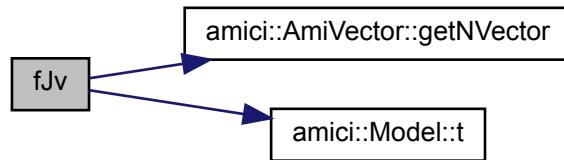
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	values of residual function (unused)
<i>v</i>	multiplication vector (unused)
<i>nJv</i>	array to which result of multiplication will be written
<i>cj</i>	scaling factor (inverse of timestep, DAE only)

Implements [Model](#).

Definition at line 48 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.10 fJv() [2/3]

```
void fJv (
    N_Vector v,
    N_Vector Jv,
    realtype t,
    N_Vector x )
```

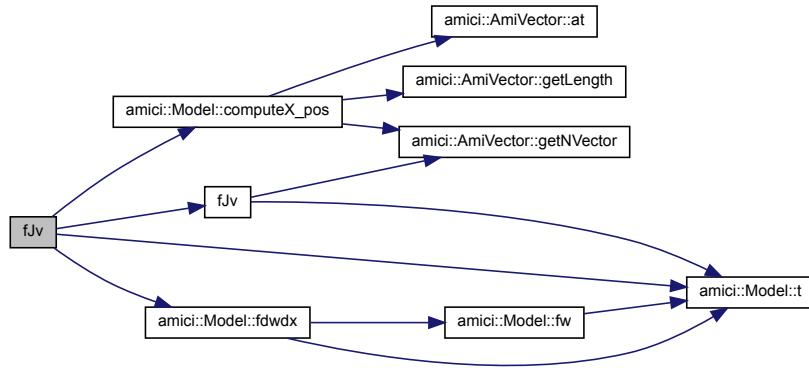
implementation of fJv at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>v</i>	Vector with which the Jacobian is multiplied
<i>Jv</i>	Vector to which the Jacobian vector product will be written

Definition at line 62 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.11 fJvB() [1/2]

```
void fJvB (
    N_Vector vB,
    N_Vector JvB,
    realtype t,
    N_Vector x,
    N_Vector xB )
```

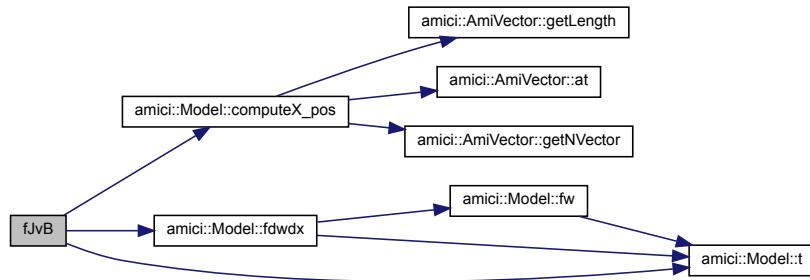
implementation of fJvB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xB</i>	Vector with the adjoint states
<i>vB</i>	Vector with which the Jacobian is multiplied
<i>JvB</i>	Vector to which the Jacobian vector product will be written

Definition at line 195 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.12 froot() [1/3]

```

void froot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    realtype * root )  [override], [virtual]
  
```

Root function

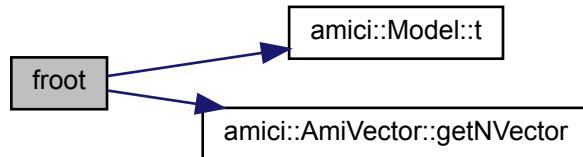
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>root</i>	array to which values of the root function will be written

Implements [Model](#).

Definition at line 70 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.13 froot() [2/3]

```
void froot (
    realtype t,
    N_Vector x,
    realtype * root )
```

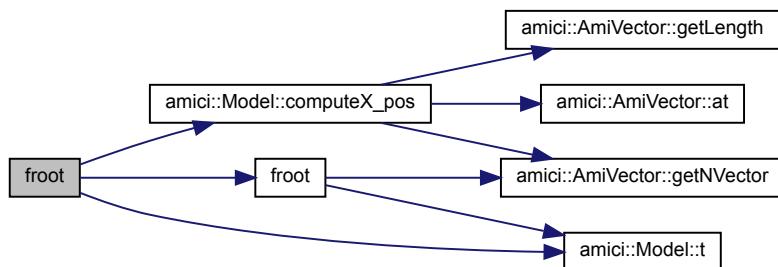
implementation of froot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>root</i>	array with root function values

Definition at line 81 of file `model_ode.cpp`.

Here is the call graph for this function:



10.13.3.14 fxdot() [1/3]

```
void fxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    AmiVector * xdot ) [override], [virtual]
```

Residual function

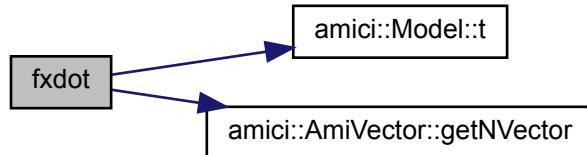
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>xdot</i>	array to which values of the residual function will be written

Implements [Model](#).

Definition at line 87 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.15 fxdot() [2/3]

```
void fxdot (
```

<pre> realtype t,</pre>	<pre> N_Vector x,</pre>
	<pre> N_Vector xdot)</pre>

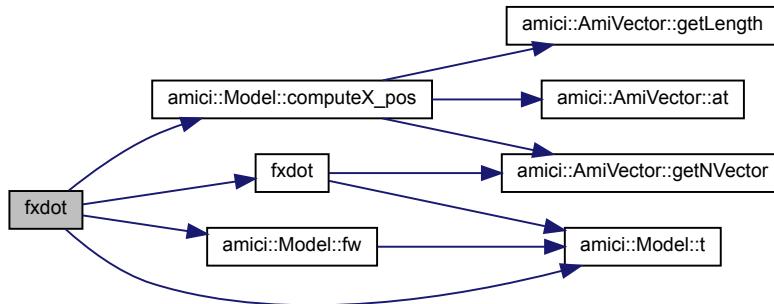
implementation of fxdot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xdot</i>	Vector with the right hand side

Definition at line 98 of file model_ode.cpp.

Here is the call graph for this function:

**10.13.3.16 fxBdot() [1/2]**

```
void fxBdot (
    realtype t,
    N_Vecor x,
    N_Vecor xB,
    N_Vecor xBdot )
```

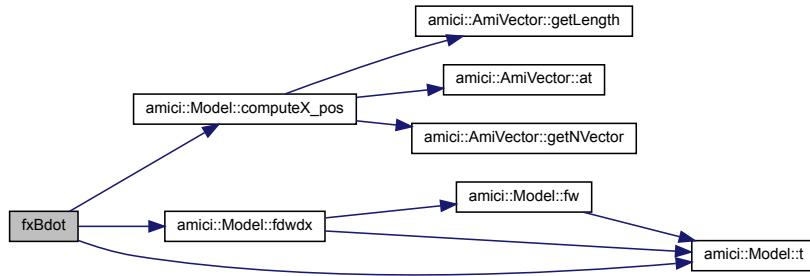
implementation of fxBdot at the N_Vecor level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xB</i>	Vector with the adjoint states
<i>xBdot</i>	Vector with the adjoint right hand side

Definition at line 210 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.17 fqBdot() [1/2]

```
void fqBdot (
    realtype t,
    N_Vector x,
    N_Vector xB,
    N_Vector qBdot )
```

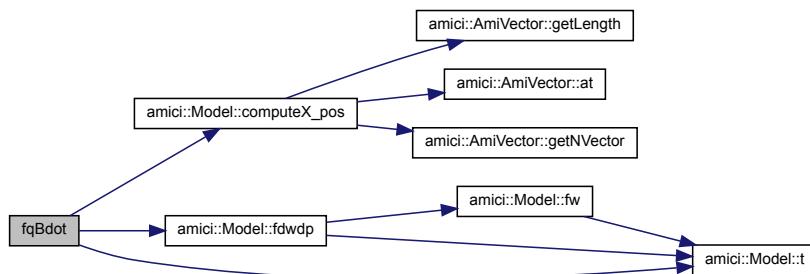
implementation of fqBdot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xB</i>	Vector with the adjoint states
<i>qBdot</i>	Vector with the adjoint quadrature right hand side

Definition at line 225 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.18 fwdxodotp() [1/3]

```
void fwdxodotp (
    const realtype t,
    const N_Vector x )
```

Sensitivity of dx/dt wrt model parameters p

Parameters

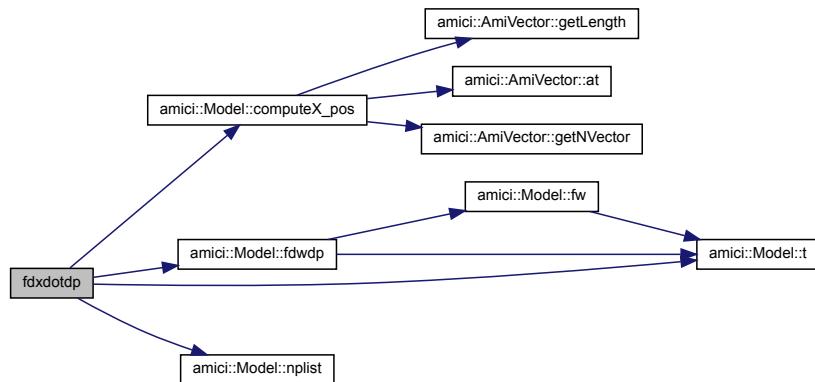
<i>t</i>	timepoint
<i>x</i>	Vector with the states

Returns

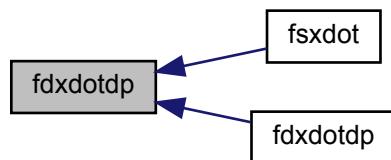
status flag indicating successful execution

Definition at line 126 of file model_ode.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.19 `fdxdotdp()` [2/3]

```
virtual void fdxdotdp (
    realtype t,
    AmiVector * x,
    AmiVector * dx ) [override], [virtual]
```

parameter derivative of residual function

Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)

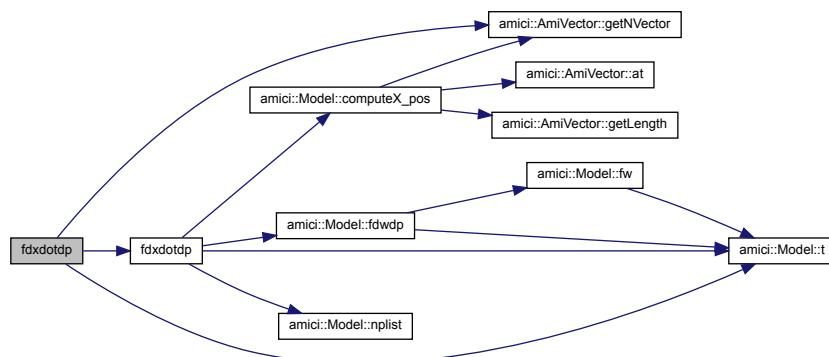
Returns

flag indicating successful evaluation

Implements [Model](#).

Definition at line 97 of file `model_ode.h`.

Here is the call graph for this function:



10.13.3.20 `fsxdot()` [1/3]

```
void fsxdot (
    realtype t,
    AmiVector * x,
    AmiVector * dx,
    int ip,
    AmiVector * sx,
    AmiVector * sdx,
    AmiVector * sxdot ) [override], [virtual]
```

Sensitivity Residual function

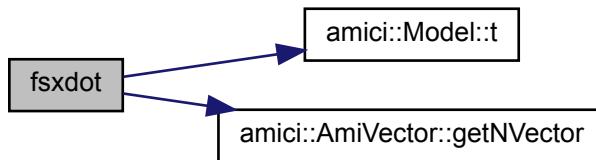
Parameters

<i>t</i>	time
<i>x</i>	state
<i>dx</i>	time derivative of state (DAE only)
<i>ip</i>	parameter index
<i>sx</i>	sensitivity state
<i>sdx</i>	time derivative of sensitivity state (DAE only)
<i>sxdot</i>	array to which values of the sensitivity residual function will be written

Implements [Model](#).

Definition at line 235 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.13.3.21 fsxdot() [2/3]

```

void fsxdot (
    realtype t,
    N_Vecor x,
    int ip,
    N_Vecor sx,
    N_Vecor sxdot )

```

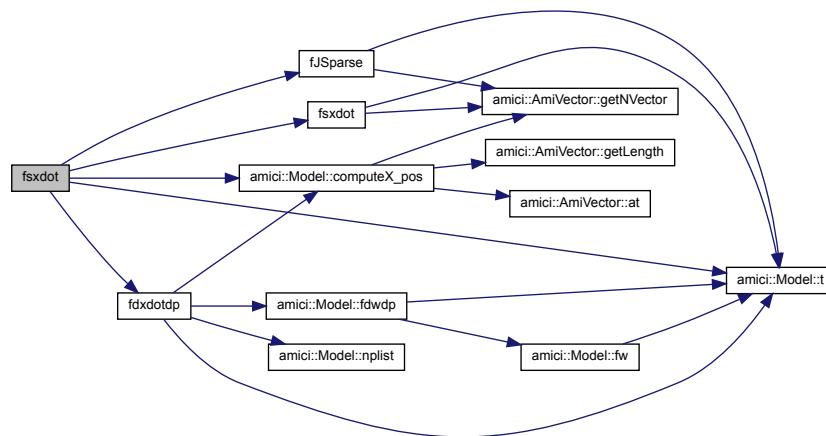
implementation of `fsxdot` at the `N_Vecor` level, this function provides an interface to the model specific routines for the solver implementation

Parameters

t	timepoint
x	Vector with the states
ip	parameter index
sx	Vector with the state sensitivities
$sxdot$	Vector with the sensitivity right hand side

Definition at line 248 of file model_ode.cpp.

Here is the call graph for this function:



10.13.3.22 getSolver()

```
std::unique_ptr< Solver > getSolver ( ) [override], [virtual]
```

Retrieves the solver object

Returns

The `Solver` instance

Implements [Model](#).

Definition at line 135 of file model_ode.cpp.

10.13.3.23 fJ() [3/3]

```
virtual void fJ (   
    realtype * J,  
    const realtype t,  
    const realtype * x,  
    const realtype * p,  
    const realtype * k,  
    const realtype * h,  
    const realtype * w,  
    const realtype * dwdx ) [protected], [pure virtual]
```

model specific implementation for f_j

Parameters

<i>J</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

10.13.3.24 fJB() [2/2]

```
virtual void fJB (
    realtype * JB,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * xB,
    const realtype * w,
    const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJB

Parameters

<i>JB</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 132 of file model_ode.h.

10.13.3.25 fJSparse() [3/3]

```
virtual void fJSparse (
    SlsMat JSparse,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
```

```
const realtype * h,
const realtype * w,
const realtype * dwdx) [protected], [pure virtual]
```

model specific implementation for fJSparse

Parameters

<i>JSparse</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

10.13.3.26 fJSparseB() [2/2]

```
virtual void fJSparseB (
    SlsMat JSparseB,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * xB,
    const realtype * w,
    const realtype * dwdx) [protected], [virtual]
```

model specific implementation for fJSparseB

Parameters

<i>JSparseB</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 160 of file model_ode.h.

10.13.3.27 fJDiag() [3/3]

```
virtual void fJDiag (
    realtype * JDiag,
```

```
const realtype t,
const realtype * x,
const realtype * p,
const realtype * k,
const realtype * h,
const realtype * w,
const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJDiag

Parameters

<i>JDiag</i>	Matrix to which the Jacobian will be written
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 175 of file model_ode.h.

10.13.3.28 fJv() [3/3]

```
virtual void fJv (
    realtype * Jv,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * v,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJv

Parameters

<i>Jv</i>	Matrix vector product of J with a vector v
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>v</i>	Vector with which the Jacobian is multiplied
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 191 of file model_ode.h.

10.13.3.29 fJvB() [2/2]

```
virtual void fJvB (
    realtype * JvB,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * xB,
    const realtype * vB,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fJvB

Parameters

<i>JvB</i>	Matrix vector product of JB with a vector v
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>vB</i>	Vector with which the Jacobian is multiplied
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 208 of file model_ode.h.

10.13.3.30 froot() [3/3]

```
virtual void froot (
    realtype * root,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [protected], [virtual]
```

model specific implementation for froot

Parameters

<i>root</i>	values of the trigger function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector

Definition at line 221 of file model_ode.h.

10.13.3.31 fxdot() [3/3]

```
virtual void fxdot (
    realtype * xdot,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * w ) [protected], [pure virtual]
```

model specific implementation for fxdot

Parameters

<i>xdot</i>	residual function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>w</i>	vector with helper variables

10.13.3.32 fxBdot() [2/2]

```
virtual void fxBdot (
    realtype * xBdot,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * xB,
    const realtype * w,
    const realtype * dwdx ) [protected], [virtual]
```

model specific implementation for fxBdot

Parameters

<i>xBdot</i>	adjoint residual function
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

Definition at line 247 of file model_ode.h.

10.13.3.33 fqBdot() [2/2]

```
virtual void fqBdot (
    realtype * qBdot,
    const int ip,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const realtype * xB,
    const realtype * w,
    const realtype * dwdp ) [protected], [virtual]
```

model specific implementation for fqBdot

Parameters

<i>qBdot</i>	adjoint quadrature equation
<i>ip</i>	sensitivity index
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>xB</i>	Vector with the adjoint states
<i>w</i>	vector with helper variables
<i>dwdp</i>	derivative of w wrt p

Definition at line 264 of file model_ode.h.

10.13.3.34 fdxdotdp() [3/3]

```
virtual void fdxdotdp (
    realtype * dxddotdp,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const realtype * w,
    const realtype * dwdp ) [protected], [virtual]
```

model specific implementation of fdxdotdp

Parameters

<i>dxddotdp</i>	partial derivative xdot wrt p
-----------------	-------------------------------

Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index
<i>w</i>	vector with helper variables
<i>dwdp</i>	derivative of w wrt p

Definition at line 280 of file model_ode.h.

10.13.3.35 fsxdot() [3/3]

```
virtual void fsxdot (
    realtype * sxdot,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h,
    const int ip,
    const realtype * sx,
    const realtype * w,
    const realtype * dwdx,
    const realtype * J,
    const realtype * dxdotdp ) [protected], [virtual]
```

model specific implementation of fsxdot

Parameters

<i>sxdot</i>	sensitivity rhs
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector
<i>h</i>	heavyside vector
<i>ip</i>	parameter index
<i>sx</i>	Vector with the state sensitivities
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x
<i>J</i>	jacobian
<i>dxdotdp</i>	parameter derivative of residual function

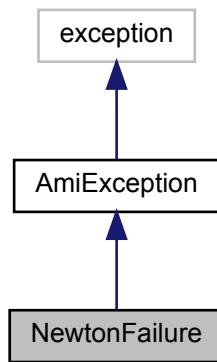
Definition at line 299 of file model_ode.h.

10.14 NewtonFailure Class Reference

newton failure exception this exception should be thrown when the steady state computation failed to converge for this exception we can assume that we can recover from the exception and return a solution struct to the user

```
#include <exception.h>
```

Inheritance diagram for NewtonFailure:



Public Member Functions

- [NewtonFailure \(int code, const char *function\)](#)

Public Attributes

- int [error_code](#)

10.14.1 Detailed Description

Definition at line 201 of file exception.h.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 NewtonFailure()

```
NewtonFailure (
    int code,
    const char * function )
```

constructor, simply calls [AmiException](#) constructor

Parameters

<i>function</i>	name of the function in which the error occurred
<i>code</i>	error code

Definition at line 209 of file exception.h.

10.14.3 Member Data Documentation

10.14.3.1 error_code

```
int error_code
```

error code returned by solver

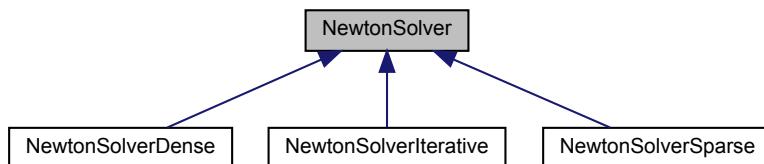
Definition at line 204 of file exception.h.

10.15 NewtonSolver Class Reference

The [NewtonSolver](#) class sets up the linear solver for the Newton method.

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

Inheritance diagram for NewtonSolver:



Public Member Functions

- [NewtonSolver \(realtype *t, AmiVector **x, Model *model, ReturnData *rdata\)](#)
- void [getStep \(int ntry, int nnewt, AmiVector *delta\)](#)
- void [computeNewtonSensis \(AmiVectorArray *sx\)](#)
- virtual void [prepareLinearSystem \(int ntry, int nnewt\)=0](#)
- virtual void [solveLinearSystem \(AmiVector *rhs\)=0](#)

Static Public Member Functions

- static std::unique_ptr<[NewtonSolver](#)> [getSolver \(realtype *t, AmiVector **x, LinearSolver linsolType, Model *model, ReturnData *rdata, int maxlinsteps, int maxsteps, double atol, double rtol\)](#)

Public Attributes

- int `maxlinsteps` = 0
- int `maxsteps` = 0
- double `atol` = 1e-16
- double `rtol` = 1e-8

Protected Attributes

- `realtypes * t`
- `Model * model`
- `ReturnData * rdata`
- `AmiVector xdot`
- `AmiVector * x`
- `AmiVector dx`

10.15.1 Detailed Description

Definition at line 25 of file `newton_solver.h`.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 NewtonSolver()

```
NewtonSolver (
    realtype * t,
    AmiVector * x,
    Model * model,
    ReturnData * rdata )
```

default constructor, initializes all members with the provided objects

Parameters

<code>t</code>	pointer to time variable
<code>x</code>	pointer to state variables
<code>model</code>	pointer to the AMICI model object
<code>rdata</code>	pointer to the return data object

Definition at line 18 of file `newton_solver.cpp`.

10.15.3 Member Function Documentation

10.15.3.1 getSolver()

```
std::unique_ptr< NewtonSolver > getSolver (
    realtype * t,
    AmiVector * x,
    LinearSolver linsolType,
    Model * model,
    ReturnData * rdata,
    int maxlinsteps,
    int maxsteps,
    double atol,
    double rtol ) [static]
```

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

Parameters

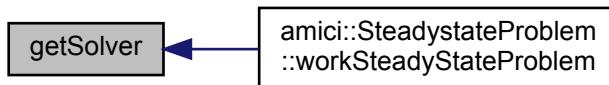
<i>t</i>	pointer to time variable
<i>x</i>	pointer to state variables
<i>linsolType</i>	integer indicating which linear solver to use
<i>model</i>	pointer to the AMICI model object
<i>rdata</i>	pointer to the return data object
<i>maxlinsteps</i>	maximum number of allowed linear steps per Newton step for steady state computation
<i>maxsteps</i>	maximum number of allowed Newton steps for steady state computation
<i>atol</i>	absolute tolerance
<i>rtol</i>	relative tolerance

Returns

solver [NewtonSolver](#) according to the specified *linsolType*

Definition at line 36 of file `newton_solver.cpp`.

Here is the caller graph for this function:



10.15.3.2 getStep()

```
void getStep (
    int ntry,
    int nnewt,
    AmiVector * delta )
```

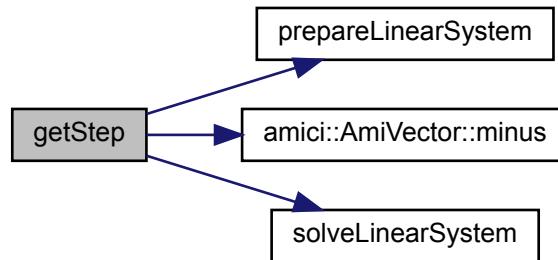
Computes the solution of one Newton iteration

Parameters

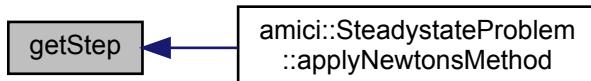
<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step
<i>delta</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system

Definition at line 107 of file newton_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.15.3.3 computeNewtonSensis()

```
void computeNewtonSensis (
    AmiVectorArray * sx )
```

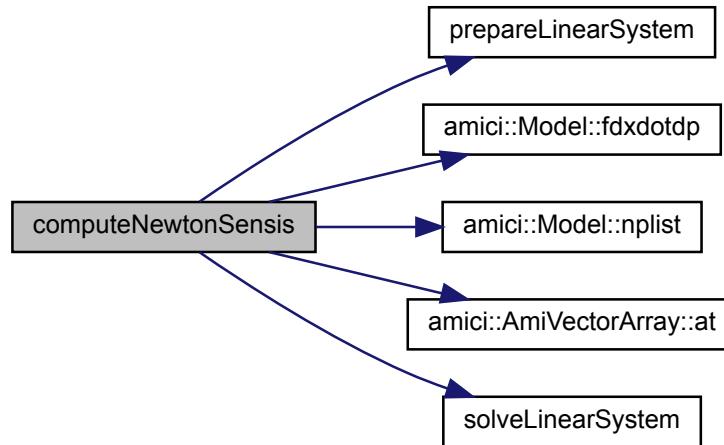
Computes steady state sensitivities

Parameters

<i>sx</i>	pointer to state variable sensitivities
-----------	---

Definition at line 127 of file newton_solver.cpp.

Here is the call graph for this function:



10.15.3.4 `prepareLinearSystem()`

```

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

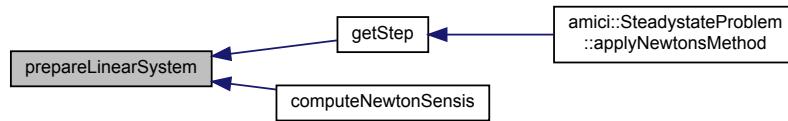
Writes the Jacobian for the Newton iteration and passes it to the linear solver

Parameters

<code>ntry</code>	integer <code>newton_try</code> integer start number of Newton solver (1 or 2)
<code>nnewt</code>	integer number of current Newton step

Implemented in [NewtonSolverIterative](#), [NewtonSolverSparse](#), and [NewtonSolverDense](#).

Here is the caller graph for this function:



10.15.3.5 solveLinearSystem()

```
virtual void solveLinearSystem (
    AmiVector * rhs ) [pure virtual]
```

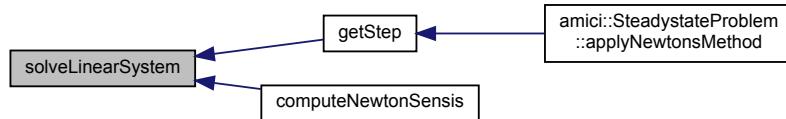
Solves the linear system for the Newton step

Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

Implemented in [NewtonSolverIterative](#), [NewtonSolverSparse](#), and [NewtonSolverDense](#).

Here is the caller graph for this function:



10.15.4 Member Data Documentation

10.15.4.1 maxlinsteps

```
int maxlinsteps = 0
```

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

Definition at line 59 of file `newton_solver.h`.

10.15.4.2 maxsteps

```
int maxsteps = 0
```

maximum number of allowed Newton steps for steady state computation

Definition at line 61 of file `newton_solver.h`.

10.15.4.3 atol

```
double atol = 1e-16
```

absolute tolerance

Definition at line 63 of file newton_solver.h.

10.15.4.4 rtol

```
double rtol = 1e-8
```

relative tolerance

Definition at line 65 of file newton_solver.h.

10.15.4.5 t

```
realtype* t [protected]
```

time variable

Definition at line 69 of file newton_solver.h.

10.15.4.6 model

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

pointer to the AMICI model object

Definition at line 71 of file newton_solver.h.

10.15.4.7 rdata

```
ReturnData* rdata [protected]
```

pointer to the return data object

Definition at line 73 of file newton_solver.h.

10.15.4.8 xdot

`AmiVector` `x` [protected]

right hand side `AmiVector`

Definition at line 75 of file `newton_solver.h`.

10.15.4.9 x

`AmiVector*` `x` [protected]

current state

Definition at line 77 of file `newton_solver.h`.

10.15.4.10 dx

`AmiVector` `dx` [protected]

current state time derivative (DAE)

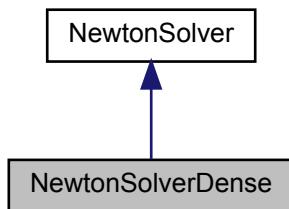
Definition at line 79 of file `newton_solver.h`.

10.16 NewtonSolverDense Class Reference

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

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

Inheritance diagram for `NewtonSolverDense`:



Public Member Functions

- `NewtonSolverDense (realtype *t, AmiVector *x, Model *model, ReturnData *rdata)`
- `void solveLinearSystem (AmiVector *rhs) override`
- `void prepareLinearSystem (int ntry, int nnewt) override`

Additional Inherited Members

10.16.1 Detailed Description

Definition at line 88 of file newton_solver.h.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 NewtonSolverDense()

```
NewtonSolverDense (
    realtype * t,
    AmiVector * x,
    Model * model,
    ReturnData * rdata )
```

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

Parameters

<code>t</code>	pointer to time variable
<code>x</code>	pointer to state variables
<code>model</code>	pointer to the AMICI model object
<code>rdata</code>	pointer to the return data object

Definition at line 152 of file newton_solver.cpp.

10.16.3 Member Function Documentation

10.16.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector * rhs ) [override], [virtual]
```

Solves the linear system for the Newton step

Parameters

<code>rhs</code>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------------	---

Solves the linear system for the Newton step

Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

Implements [NewtonSolver](#).

Definition at line 191 of file `newton_solver.cpp`.

Here is the call graph for this function:



10.16.3.2 `prepareLinearSystem()`

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

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

Parameters

<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step

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

Parameters

<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step

Implements [NewtonSolver](#).

Definition at line 171 of file `newton_solver.cpp`.

Here is the call graph for this function:

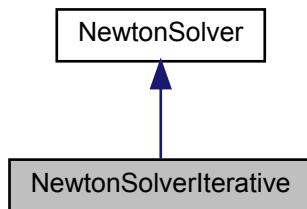


10.17 NewtonSolverIterative Class Reference

The [NewtonSolverIterative](#) provides access to the iterative linear solver for the Newton method.

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

Inheritance diagram for NewtonSolverIterative:



Public Member Functions

- [NewtonSolverIterative \(realtype *t, AmiVector *x, Model *model, ReturnData *rdata\)](#)
- void [solveLinearSystem \(AmiVector *rhs\)](#)
- void [prepareLinearSystem \(int ntry, int nnewt\)](#)
- void [linsolveSPBCG \(int ntry, int nnewt, AmiVector *ns_delta\)](#)

Additional Inherited Members

10.17.1 Detailed Description

Definition at line 136 of file `newton_solver.h`.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 NewtonSolverIterative()

```
NewtonSolverIterative (
    realtype * t,
    AmiVector * x,
    Model * model,
    ReturnData * rdata )
```

default constructor, initializes all members with the provided objects

Parameters

<i>t</i>	pointer to time variable
<i>x</i>	pointer to state variables
<i>model</i>	pointer to the AMICI model object
<i>rdata</i>	pointer to the return data object

Definition at line 312 of file newton_solver.cpp.

10.17.3 Member Function Documentation

10.17.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector * rhs ) [virtual]
```

Solves the linear system for the Newton step

Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

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

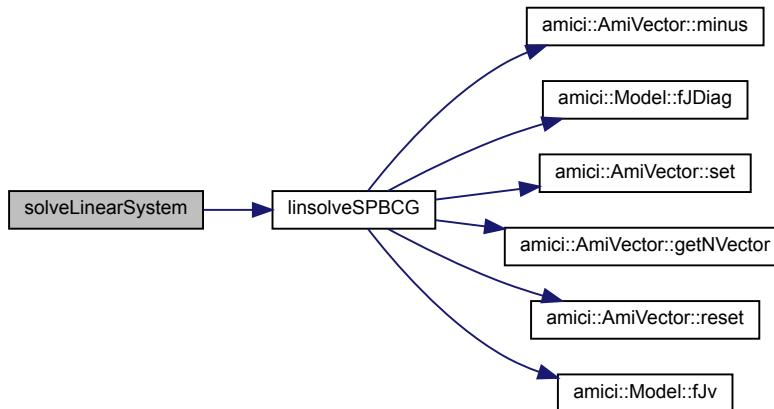
Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

Implements [NewtonSolver](#).

Definition at line 350 of file newton_solver.cpp.

Here is the call graph for this function:



10.17.3.2 prepareLinearSystem()

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

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

Parameters

<code>ntry</code>	integer newton_try integer start number of Newton solver (1 or 2)
<code>nnewt</code>	integer number of current Newton step

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

Parameters

<code>ntry</code>	integer newton_try integer start number of Newton solver (1 or 2)
<code>nnewt</code>	integer number of current Newton step

Implements [NewtonSolver](#).

Definition at line 329 of file `newton_solver.cpp`.

10.17.3.3 linsolveSPBCG()

```
void linsolveSPBCG (
    int ntry,
```

```
int nnewt,
AmiVector * ns_delta )
```

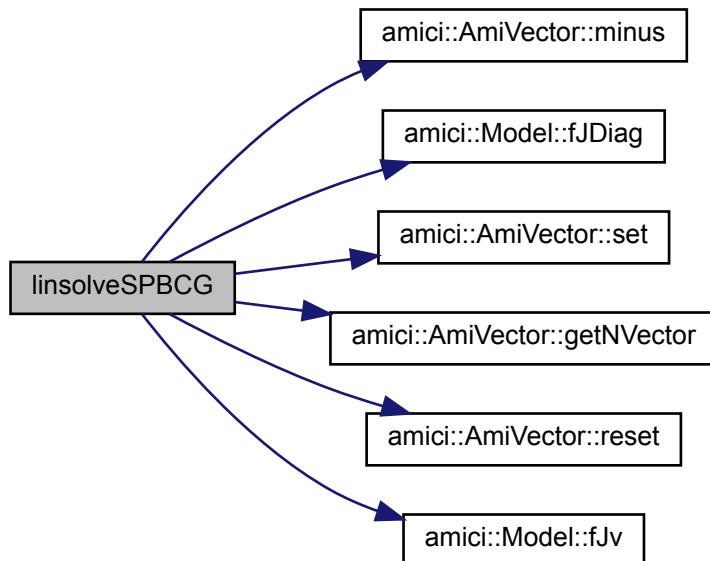
Iterative linear solver created from SPILS BiCG-Stab. Solves the linear system within each Newton step if iterative solver is chosen.

Parameters

<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step
<i>ns_delta</i>	Newton step

Definition at line 363 of file newton_solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

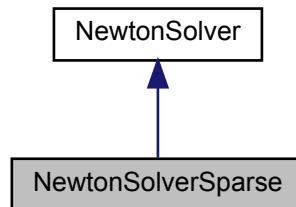


10.18 NewtonSolverSparse Class Reference

The [NewtonSolverSparse](#) provides access to the sparse linear solver for the Newton method.

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

Inheritance diagram for NewtonSolverSparse:



Public Member Functions

- [NewtonSolverSparse \(realtype *t, AmiVector *x, Model *model, ReturnData *rdata\)](#)
- void [solveLinearSystem \(AmiVector *rhs\)](#) override
- void [prepareLinearSystem \(int ntry, int nnewt\)](#) override

Additional Inherited Members

10.18.1 Detailed Description

Definition at line 109 of file `newton_solver.h`.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 NewtonSolverSparse()

```
NewtonSolverSparse (
    realtype * t,
    AmiVector * x,
    Model * model,
    ReturnData * rdata )
```

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

Parameters

<i>t</i>	pointer to time variable
<i>x</i>	pointer to state variables
<i>model</i>	pointer to the AMICI model object
<i>rdata</i>	pointer to the return data object

Definition at line 221 of file newton_solver.cpp.

10.18.3 Member Function Documentation

10.18.3.1 solveLinearSystem()

```
void solveLinearSystem (
    AmiVector * rhs ) [override], [virtual]
```

Solves the linear system for the Newton step

Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

Solves the linear system for the Newton step

Parameters

<i>rhs</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system
------------	---

Implements [NewtonSolver](#).

Definition at line 278 of file newton_solver.cpp.

Here is the call graph for this function:



10.18.3.2 `prepareLinearSystem()`

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

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

Parameters

<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step

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

Parameters

<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step

Implements [NewtonSolver](#).

Definition at line 244 of file `newton_solver.cpp`.

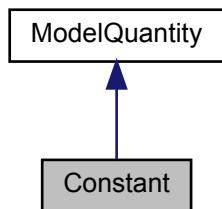
Here is the call graph for this function:



10.19 Constant Class Reference

A [Constant](#) is a fixed variable in the model with respect to which sensitivities cannot be computed, abbreviated by `k`

Inheritance diagram for Constant:



Public Member Functions

- def `__init__` (self, identifier, name, value)
Create a new Expression instance.

10.19.1 Detailed Description

Definition at line 496 of file `ode_export.py`.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

Parameters

<code>identifier</code>	unique identifier of the Constant Type: sympy.Symbol
<code>name</code>	individual name of the Constant (does not need to be unique) Type: str
<code>value</code>	numeric value Type: float

Returns

`ModelQuantity` instance

`TypeError`

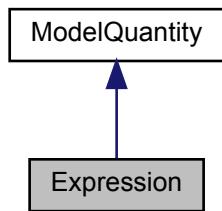
is thrown if input types do not match documented types

Definition at line 515 of file `ode_export.py`.

10.20 Expression Class Reference

An Expressions is a recurring elements in symbolic formulas.

Inheritance diagram for Expression:



Public Member Functions

- def `__init__` (self, identifier, name, value)
Create a new [Expression](#) instance.

10.20.1 Detailed Description

Specifying this may yield more compact expression which may lead to substantially shorter model compilation times, but may also reduce model simulation time, abbreviated by `w`

Definition at line 443 of file `ode_export.py`.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

Parameters

<code>identifier</code>	unique identifier of the Expression Type: <code>sympy.Symbol</code>
<code>name</code>	individual name of the Expression (does not need to be unique) Type: <code>str</code>
<code>value</code>	formula Type: <code>symengine.Basic</code>

Returns

[ModelQuantity](#) instance

TypeError

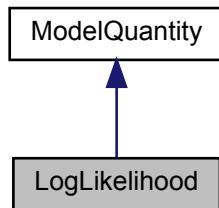
is thrown if input types do not match documented types

Definition at line 461 of file `ode_export.py`.

10.21 LogLikelihood Class Reference

A [LogLikelihood](#) defines the distance between measurements and experiments for a particular observable.

Inheritance diagram for LogLikelihood:

**Public Member Functions**

- def [`__init__`](#) (self, identifier, name, value)

Create a new [Expression](#) instance.

10.21.1 Detailed Description

The final [LogLikelihood](#) value in the simulation will be the sum of all specified [LogLikelihood](#) instances evaluated at all timepoints, abbreviated by JY

Definition at line 525 of file `ode_export.py`.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 `__init__()`

```

def __init__ (
    self,
    identifier,
    name,
    value )
  
```

Parameters

<i>identifier</i>	unique identifier of the LogLikelihood Type: sympy.Symbol
<i>name</i>	individual name of the LogLikelihood (does not need to be unique) Type: str
<i>value</i>	formula Type: symengine.Basic

Returns

[ModelQuantity](#) instance

TypeError

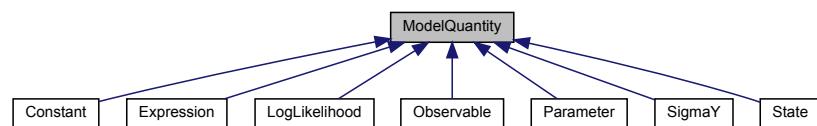
is thrown if input types do not match documented types

Definition at line 545 of file `ode_export.py`.

10.22 ModelQuantity Class Reference

Base class for model components.

Inheritance diagram for ModelQuantity:

**Public Member Functions**

- def [`__init__`](#) (self, identifier, name, value)
Create a new [ModelQuantity](#) instance.
- def [`__repr__`](#) (self)
Representation of the [ModelQuantity](#) object.

10.22.1 Detailed Description

Definition at line 289 of file `ode_export.py`.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 __init__()

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

This function sanitizes input from pysb to make sure we are operating on flat symby.Symbol and sympy.Basic and not respective derived pysb classes

Parameters

<i>identifier</i>	unique identifier of the quantity Type: sympy.Symbol
<i>name</i>	individual name of the quantity (does not need to be unique) Type: str
<i>value</i>	either formula, numeric value or initial value

Returns

[ModelQuantity](#) instance

TypeError

is thrown if input types do not match documented types

Definition at line 309 of file `ode_export.py`.

Here is the call graph for this function:



10.22.3 Member Function Documentation

10.22.3.1 __repr__()

```
def __repr__ (
```

self)

Returns

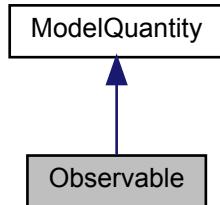
string representation of the [ModelQuantity](#)

Definition at line 348 of file `ode_export.py`.

10.23 Observable Class Reference

An [Observable](#) links model simulations to experimental measurements, abbreviated by `y`

Inheritance diagram for Observable:



Public Member Functions

- def `__init__` (self, identifier, name, value)
Create a new Observable instance.

10.23.1 Detailed Description

Definition at line 389 of file `ode_export.py`.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

Parameters

<code>identifier</code>	unique identifier of the Observable Type: <code>sympy.Symbol</code>
<code>name</code>	individual name of the Observable (does not need to be unique) Type: <code>str</code>
<code>value</code>	formula Type: <code>symengine.Basic</code>

Returns

[ModelQuantity](#) instance

TypeError

is thrown if input types do not match documented types

Definition at line 407 of file `ode_export.py`.

10.24 ODEExporter Class Reference

The [ODEExporter](#) class generates AMICI C++ files for ODE model as defined in symbolic expressions.

Public Member Functions

- def [__init__](#) (self, `ode_model`, `outdir=None`, `verbose=False`, `assume_pow_positivity=False`, `compiler=None`)
Generate AMICI C++ files for the ODE provided to the constructor.
- def [generateModelCode](#) (self)
Generates the native C++ code for the loaded model.
- def [compileModel](#) (self)
Compiles the generated code it into a simulatable module.
- def [setPaths](#) (self, `output_dir`)
Set output paths for the model and create if necessary.
- def [setName](#) (self, `modelName`)
Sets the model name.

Public Attributes

- `outdir`
see `sbml_import.setPaths()`
Type: str
- `verbose`
more verbose output if True
Type: bool
- `assume_pow_positivity`
if set to true, a special pow function is
- `compiler`
distutils/setuptools compiler selection to build the
- `modelName`
name of the model that will be used for compilation
- `modelPath`
path to the generated model specific files
Type: str
- `modelSwigPath`
path to the generated swig files
Type: str
- `model`
ODE definition
Type: [ODEModel](#).
- `functions`
carries C++ function signatures and other specifications
- `allow_reinit_fixpar_initcond`
indicates whether reinitialization of

10.24.1 Detailed Description

initial states depending on fixedParameters is allowed for this model

Type: bool

Definition at line 1480 of file `ode_export.py`.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 `__init__()`

```
def __init__ (
    self,
    ode_model,
    outdir = None,
    verbose = False,
    assume_pow_positivity = False,
    compiler = None )
```

Parameters

<code>ode_model</code>	ODE definition Type: <code>ODEModel</code>
<code>outdir</code>	see <code>sbml_import.setPaths()</code> Type: str
<code>verbose</code>	more verbose output if True Type: bool
<code>assume_pow_positivity</code>	if set to true, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors Type: bool
<code>compiler</code>	distutils/setuptools compiler selection to build the python extension Type: str

Returns

Definition at line 1568 of file `ode_export.py`.

10.24.3 Member Function Documentation

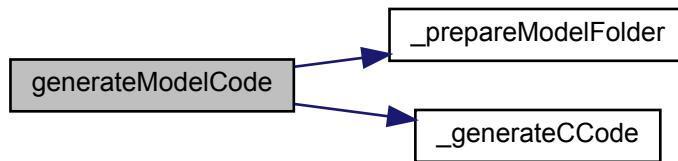
10.24.3.1 `generateModelCode()`

```
def generateModelCode (
    self )
```

Returns

Definition at line 1600 of file `ode_export.py`.

Here is the call graph for this function:

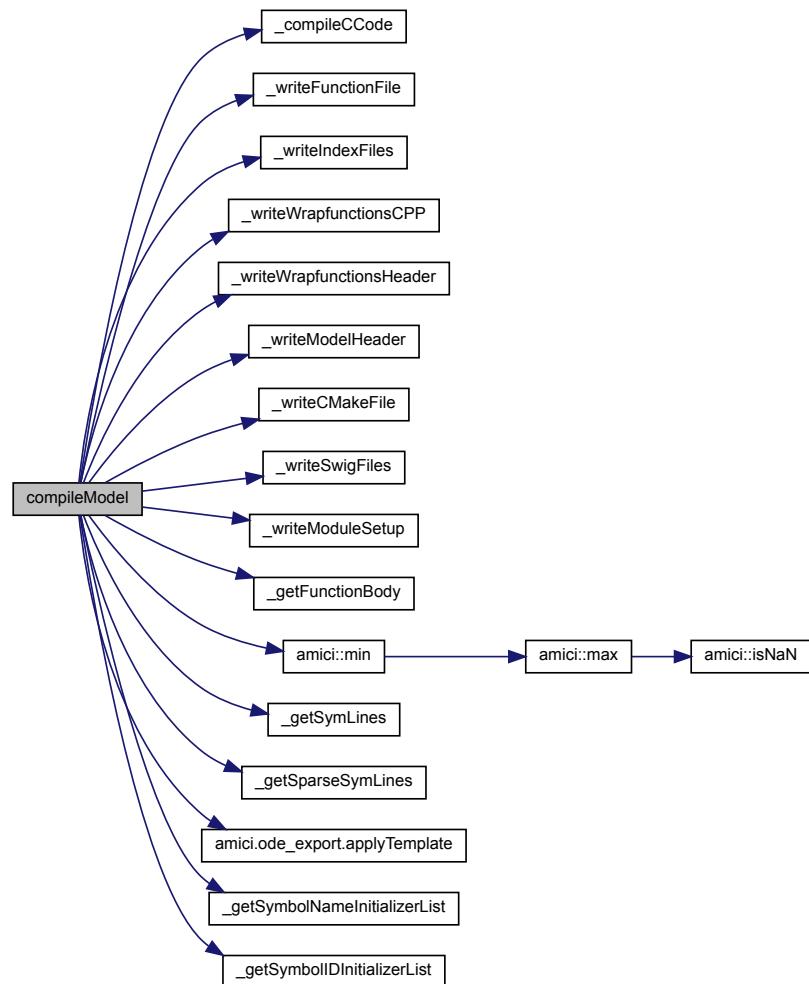
**10.24.3.2 compileModel()**

```
def compileModel ( self )
```

Returns

Definition at line 1613 of file `ode_export.py`.

Here is the call graph for this function:



10.24.3.3 setPaths()

```
def setPaths ( self, output_dir )
```

Parameters

<code>output_dir</code>	relative or absolute path where the generated model code is to be placed. will be created if does not exists. Type: str
-------------------------	---

Returns

Definition at line 2201 of file `ode_export.py`.

10.24.3.4 setName()

```
def setName (
    self,
    modelName )
```

Parameters

<code>modelName</code>	name of the model (must only contain valid filename characters) Type: str
------------------------	---

Returns

Definition at line 2220 of file `ode_export.py`.

10.25 ODEModel Class Reference

An `ODEModel` defines an Ordinary Differential Equation as set of `ModelQuantities`.

Public Member Functions

- def `__init__` (`self`)

Create a new `ODEModel` instance.
- def `import_from_sbml_importer` (`self, si`)

Imports a model specification from a `amici.SBMLImporter` instance.
- def `add_component` (`self, component`)

Adds a new `ModelQuantity` to the model.
- def `nx` (`self`)

Number of states.
- def `ny` (`self`)

Number of Observables.
- def `nk` (`self`)

Number of Constants.

- def `np` (`self`)

Number of Parameters.
- def `sym` (`self, name`)

Returns (and constructs if necessary) the identifiers for a symbolic entity.
- def `sparsesym` (`self, name`)

Returns (and constructs if necessary) the sparsified identifiers for a sparsified symbolic variable.
- def `eq` (`self, name`)

Returns (and constructs if necessary) the formulas for a symbolic entity.
- def `sparseeq` (`self, name`)

Returns (and constructs if necessary) the sparsified formulas for a sparsified symbolic variable.
- def `colptr` (`self, name`)

Returns (and constructs if necessary) the column pointers for a sparsified symbolic variable.
- def `rowval` (`self, name`)

Returns (and constructs if necessary) the row values for a sparsified symbolic variable.
- def `val` (`self, name`)

Returns (and constructs if necessary) the numeric values of a symbolic entity.
- def `name` (`self, name`)

Returns (and constructs if necessary) the names of a symbolic variable.
- def `generateBasicVariables` (`self`)

Generates the symbolic identifiers for all variables in `ODEModel.variable_prototype`.
- def `symNames` (`self`)

Returns a list of names of generated symbolic variables.

10.25.1 Detailed Description

This class provides general purpose interfaces to compute arbitrary symbolic derivatives that are necessary for model simulation or sensitivity computation

derivative from a partial derivative call to enforce a partial derivative in the next recursion. prevents infinite recursion

Definition at line 570 of file `ode_export.py`.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 `__init__()`

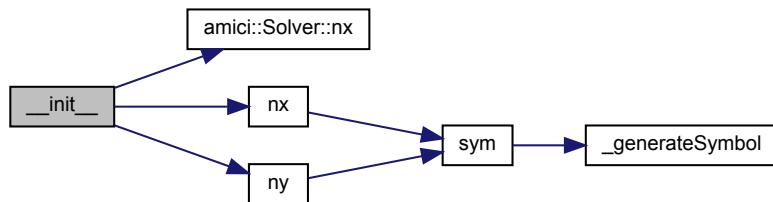
```
def __init__ (
    self )
```

Returns

New [ODEModel](#) instance

Definition at line 725 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3 Member Function Documentation

10.25.3.1 import_from_sbml_importer()

```
def import_from_sbml_importer (
    self,
    si )
```

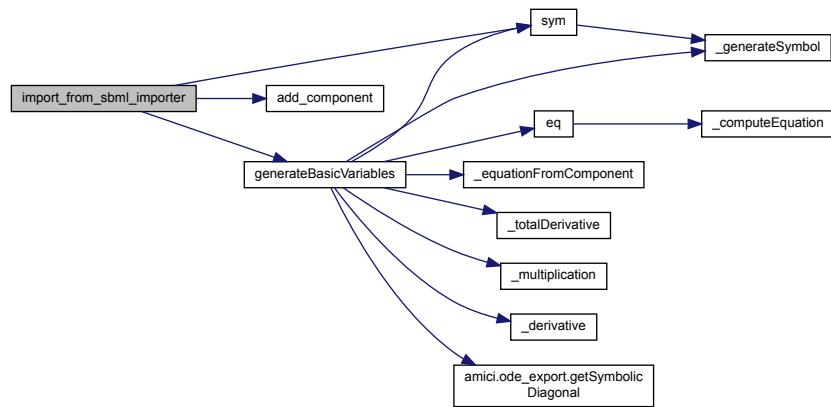
Parameters

<i>si</i>	imported SBML model
	Type: amici.SbmlImporter

Returns

Definition at line 818 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3.2 add_component()

```
def add_component (
    self,
    component )
```

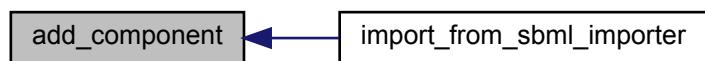
Parameters

<i>component</i>	model quantity to be added Type: ModelQuantity
------------------	---

Returns

Definition at line 851 of file `ode_export.py`.

Here is the caller graph for this function:



10.25.3.3 nx()

```
def nx ( self )
```

Returns

number of state variable symbols

Definition at line 871 of file ode_export.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.25.3.4 ny()

```
def ny ( self )
```

Returns

number of observable symbols

Definition at line 884 of file ode_export.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.25.3.5 nk()

```
def nk (self )
```

Returns

number of constant symbols

Definition at line 897 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3.6 np()

```
def np ( self )
```

Returns

number of parameter symbols

Definition at line 910 of file ode_export.py.

Here is the call graph for this function:



10.25.3.7 sym()

```
def sym ( self, name )
```

Parameters

<i>name</i>	name of the symbolic variable
Type:	str

Returns

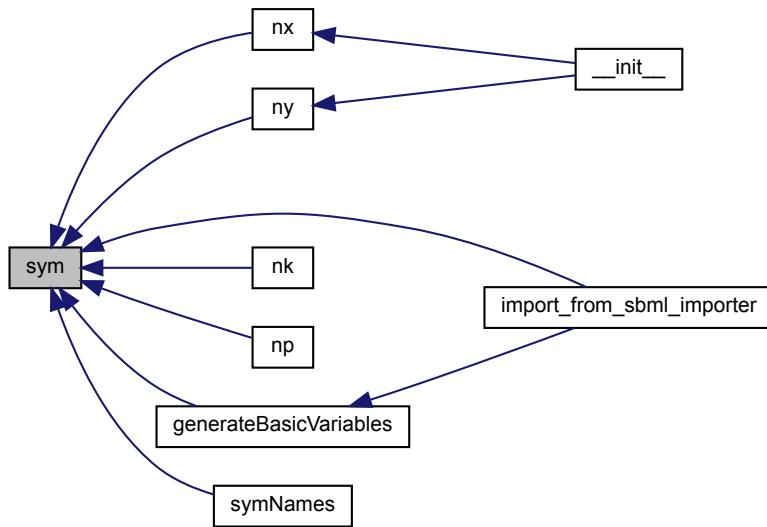
containing the symbolic identifiers
Type: symengine.DenseMatrix

Definition at line 925 of file ode_export.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.25.3.8 sparsesym()

```
def sparsesym (
    self,
    name )
```

Parameters

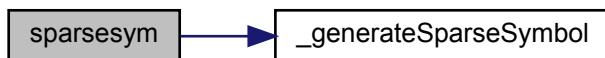
<code>name</code>	name of the symbolic variable Type: str
-------------------	---

Returns

linearized symengine.DenseMatrix containing the symbolic identifiers

Definition at line 942 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3.9 eq()

```
def eq ( self, name )
```

Parameters

<i>name</i>	name of the symbolic variable
	Type: str

Returns

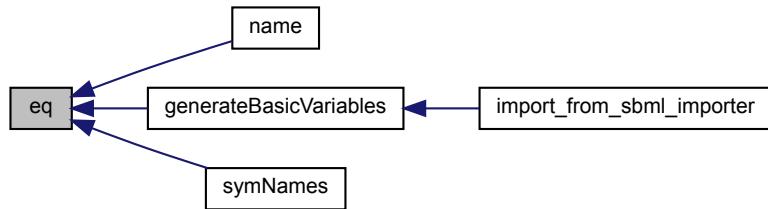
symengine.DenseMatrix containing the symbolic identifiers

Definition at line 961 of file ode_export.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.25.3.10 sparseeq()

```
def sparseeq ( self, name )
```

Parameters

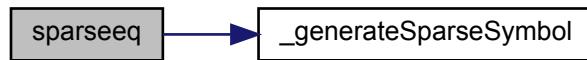
<i>name</i>	name of the symbolic variable Type: str
-------------	---

Returns

linearized symengine.DenseMatrix containing the symbolic formulas

Definition at line 978 of file `ode_export.py`.

Here is the call graph for this function:

**10.25.3.11 colptr()**

```
def colptr (
    self,
    name )
```

Parameters

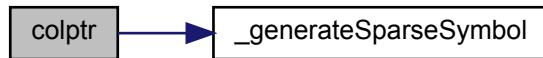
<i>name</i>	name of the symbolic variable Type: str
-------------	---

Returns

symengine.DenseMatrix containing the column pointers

Definition at line 997 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3.12 rowval()

```
def rowval (
    self,
    name )
```

Parameters

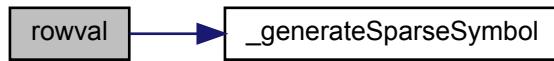
<i>name</i>	name of the symbolic variable
	Type: str

Returns

`symengine.DenseMatrix` containing the row values

Definition at line 1016 of file `ode_export.py`.

Here is the call graph for this function:

**10.25.3.13 val()**

```
def val (
    self,
    name )
```

Parameters

<i>name</i>	name of the symbolic variable
	Type: str

Returns

list containing the numeric values

Definition at line 1035 of file `ode_export.py`.

Here is the call graph for this function:



10.25.3.14 name()

```
def name ( self, name )
```

Parameters

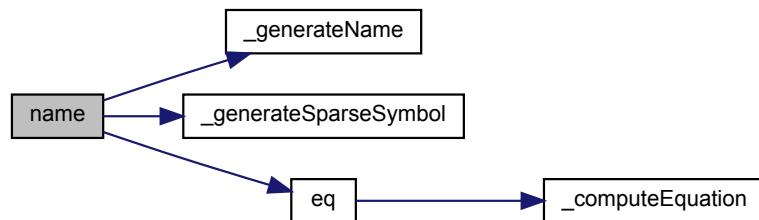
<i>name</i>	name of the symbolic variable Type: str
-------------	---

Returns

list of names

Definition at line 1051 of file `ode_export.py`.

Here is the call graph for this function:



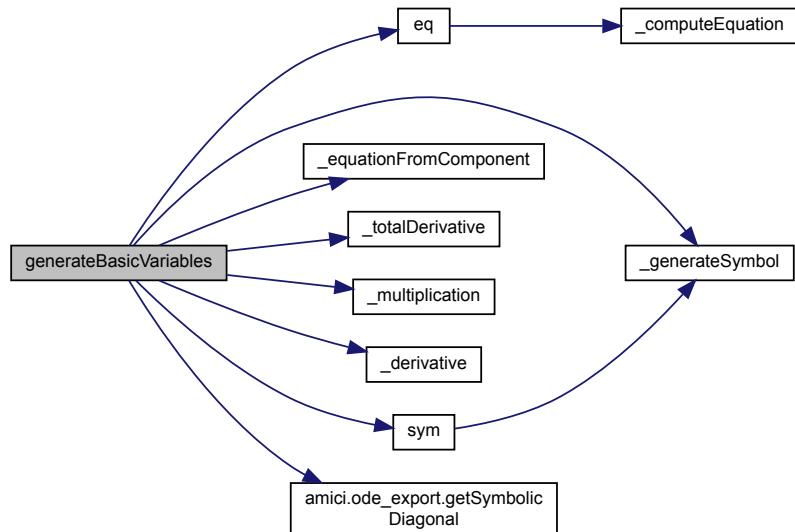
10.25.3.15 generateBasicVariables()

```
def generateBasicVariables (
    self )
```

Returns

Definition at line 1106 of file `ode_export.py`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.25.3.16 symNames()

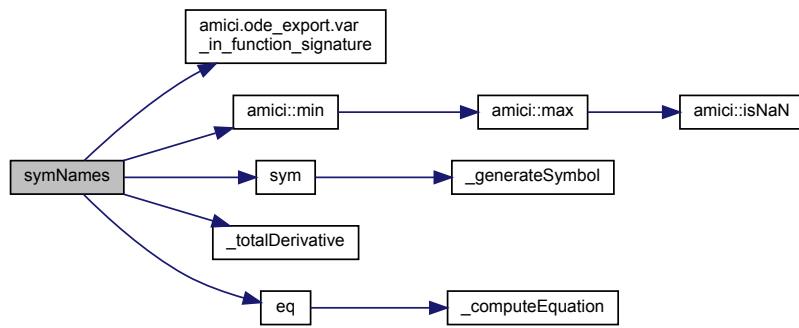
```
def symNames (
    self )
```

Returns

list of names

Definition at line 1255 of file `ode_export.py`.

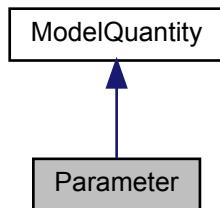
Here is the call graph for this function:



10.26 Parameter Class Reference

A [Parameter](#) is a free variable in the model with respect to which sensitivities may be computed, abbreviated by `p`

Inheritance diagram for Parameter:



Public Member Functions

- def `__init__` (self, identifier, name, value)
Create a new [Expression](#) instance.

10.26.1 Detailed Description

Definition at line 469 of file `ode_export.py`.

10.26.2 Constructor & Destructor Documentation

10.26.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

Parameters

<code>identifier</code>	unique identifier of the Parameter Type: <code>sympy.Symbol</code>
<code>name</code>	individual name of the Parameter (does not need to be unique) Type: <code>str</code>
<code>value</code>	numeric value Type: <code>float</code>

Returns

[ModelQuantity](#) instance

`TypeError`

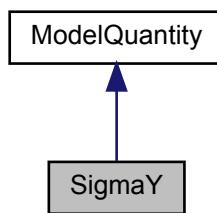
is thrown if input types do not match documented types

Definition at line 488 of file `ode_export.py`.

10.27 SigmaY Class Reference

A Standard Deviation [SigmaY](#) rescales the distance between simulations and measurements when computing residuals, abbreviated by `sigmay`

Inheritance diagram for SigmaY:



Public Member Functions

- def `__init__` (self, identifier, name, value)
Create a new Standard Deviation instance.

10.27.1 Detailed Description

Definition at line 415 of file `ode_export.py`.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value )
```

Parameters

<code>identifier</code>	unique identifier of the Standard Deviation Type: <code>sympy.Symbol</code>
<code>name</code>	individual name of the Standard Deviation (does not need to be unique) Type: <code>str</code>
<code>value</code>	formula Type: <code>symengine.Basic</code>

Returns

`ModelQuantity` instance

`TypeError`

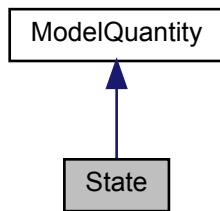
is thrown if input types do not match documented types

Definition at line 434 of file `ode_export.py`.

10.28 State Class Reference

A `State` variable defines an entity that evolves with time according to the provided time derivative, abbreviated by `x`

Inheritance diagram for State:



Public Member Functions

- def `__init__` (self, identifier, name, value, dt)
Create a new [State](#) instance.

10.28.1 Detailed Description

Definition at line 356 of file `ode_export.py`.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 `__init__()`

```
def __init__ (
    self,
    identifier,
    name,
    value,
    dt )
```

Extends [ModelQuantity.__init__](#) by dt

Parameters

<code>identifier</code>	unique identifier of the state Type: <code>sympy.Symbol</code>
<code>name</code>	individual name of the state (does not need to be unique) Type: <code>str</code>
<code>value</code>	initial value Type: <code>symengine.Basic</code>
<code>dt</code>	time derivative Type: <code>symengine.Basic</code>

Returns

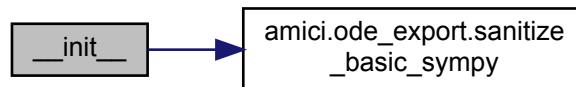
[ModelQuantity](#) instance

TypeError

is thrown if input types do not match documented types

Definition at line 377 of file `ode_export.py`.

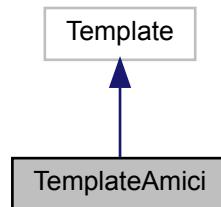
Here is the call graph for this function:



10.29 TemplateAmici Class Reference

Template format used in AMICI (see `string.Template` for more details).

Inheritance diagram for `TemplateAmici`:

**Static Public Attributes**

- string `delimiter` = 'TPL_'
delimiter that identifies template variables
Type: str

10.29.1 Detailed Description

Definition at line 2248 of file `ode_export.py`.

10.30 ReturnData Class Reference

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

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

Public Member Functions

- `ReturnData ()`
default constructor
- `ReturnData (std::vector< realtype > ts, int np, int nk, int nx, int nxtrue, int ny, int nytrue, int nz, int nztrue, int ne, int nJ, int nplist, int nmaxevent, int nt, int newton_maxsteps, std::vector< ParameterScaling > pscale, SecondOrderMode o2mode, SensitivityOrder sensi, SensitivityMethod sensi_meth)`
ReturnData.
- `ReturnData (Solver const &solver, const Model *model)`
- `void initializeObjectiveFunction ()`
initializeObjectiveFunction
- `void invalidate (const realtype t)`
- `void invalidateLLH ()`
- `void applyChainRuleFactorToSimulationResults (const Model *model)`

Public Attributes

- `const std::vector< realtype > ts`
- `std::vector< realtype > xdot`
- `std::vector< realtype > J`
- `std::vector< realtype > z`
- `std::vector< realtype > sigmaz`
- `std::vector< realtype > sz`
- `std::vector< realtype > ssigmaz`
- `std::vector< realtype > rz`
- `std::vector< realtype > srz`
- `std::vector< realtype > s2rz`
- `std::vector< realtype > x`
- `std::vector< realtype > sx`
- `std::vector< realtype > y`
- `std::vector< realtype > sigmay`
- `std::vector< realtype > sy`
- `std::vector< realtype > ssigmay`
- `std::vector< realtype > res`
- `std::vector< realtype > sres`
- `std::vector< realtype > FIM`
- `std::vector< int > numsteps`
- `std::vector< int > numstepsB`
- `std::vector< int > numrhsevals`
- `std::vector< int > numrhsevalsB`
- `std::vector< int > numerptestfails`
- `std::vector< int > numerptestfailsB`
- `std::vector< int > numnonlinsolvconvfails`
- `std::vector< int > numnonlinsolvconvfailsB`
- `std::vector< int > order`
- `int newton_status = 0`

- double newton_cpu_time = 0.0
- std::vector< int > newton_numsteps
- std::vector< int > newton_numlinsteps
- realtype t_steadystate = NAN
- realtype wrms_steadystate = NAN
- realtype wrms_sensi_steadystate = NAN
- std::vector< realtype > x0
- std::vector< realtype > sx0
- realtype llh = 0.0
- realtype chi2 = 0.0
- std::vector< realtype > sllh
- std::vector< realtype > s2llh
- int status = 0
- 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
- std::vector< ParameterScaling > pscale
- const SecondOrderMode o2mode
- const SensitivityOrder sensi
- const SensitivityMethod sensi_meth

Friends

- template<class Archive >
void boost::serialization::serialize (Archive &ar, ReturnData &r, const unsigned int version)
Serialize ReturnData (see boost::serialization::serialize)

10.30.1 Detailed Description

NOTE: multidimensional arrays are stored in row-major order (FORTRAN-style)

Definition at line 28 of file rdata.h.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 ReturnData() [1/2]

```
ReturnData (
    std::vector< realtype > ts,
    int np,
    int nk,
    int nx,
    int nxtrue,
    int ny,
    int nytrue,
    int nz,
    int nztrue,
    int ne,
    int nJ,
    int nplist,
    int nmaxevent,
    int nt,
    int newton_maxsteps,
    std::vector< ParameterScaling > pscale,
    SecondOrderMode o2mode,
    SensitivityOrder sensi,
    SensitivityMethod sensi_meth )
```

Parameters

<i>ts</i>	see amici::Model::ts
<i>np</i>	see amici::Model::np
<i>nk</i>	see amici::Model::nk
<i>nx</i>	see amici::Model::nx
<i>nxtrue</i>	see amici::Model::nxtrue
<i>ny</i>	see amici::Model::ny
<i>nytrue</i>	see amici::Model::nytrue
<i>nz</i>	see amici::Model::nz
<i>nztrue</i>	see amici::Model::nztrue
<i>ne</i>	see amici::Model::ne
<i>nJ</i>	see amici::Model::nJ
<i>nplist</i>	see amici::Model::nplist
<i>nmaxevent</i>	see amici::Model::nmaxevent
<i>nt</i>	see amici::Model::nt
<i>newton_maxsteps</i>	see amici::Solver::newton_maxsteps
<i>pscale</i>	see amici::Model::pscale
<i>o2mode</i>	see amici::Model::o2mode
<i>sensi</i>	see amici::Solver::sensi
<i>sensi_meth</i>	see amici::Solver::sensi_meth

Definition at line 40 of file rdata.cpp.

Here is the call graph for this function:



10.30.2.2 ReturnData() [2/2]

```
ReturnData (
    Solver const & solver,
    const Model * model )
```

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

Parameters

solver	solver
model	pointer to model specification object bool

Definition at line 22 of file rdata.cpp.

10.30.3 Member Function Documentation

10.30.3.1 invalidate()

```
void invalidate (
    const realltype t )
```

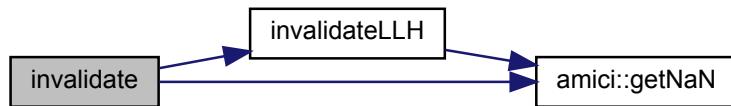
routine to set likelihood, state variables, outputs and respective sensitivities to NaN (typically after integration failure)

Parameters

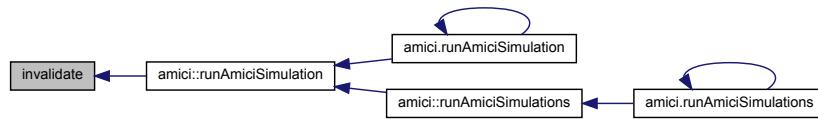
t	time of integration failure
---	-----------------------------

Definition at line 117 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.30.3.2 invalidateLLH()

```
void invalidateLLH ( )
```

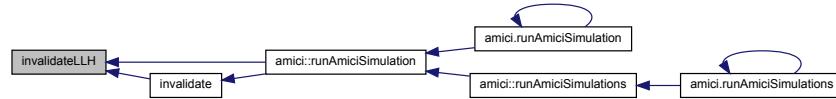
routine to set likelihood and respective sensitivities to NaN (typically after integration failure)

Definition at line 156 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.30.3.3 applyChainRuleFactorToSimulationResults()

```
void applyChainRuleFactorToSimulationResults (
    const Model * model )
```

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

Parameters

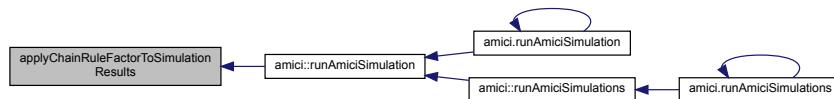
<code>model</code>	Model from which the ReturnData was obtained
--------------------	--

Definition at line 168 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.30.4 Friends And Related Function Documentation

10.30.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
    Archive & ar,
    ReturnData & r,
    const unsigned int version ) [friend]
```

Parameters

<code>ar</code>	Archive to serialize to
<code>r</code>	Data to serialize
<code>version</code>	Version number

10.30.5 Member Data Documentation

10.30.5.1 ts

```
const std::vector<realtyp> ts  
timepoints (dimension: nt)
```

Definition at line 77 of file rdata.h.

10.30.5.2 xdot

```
std::vector<realtyp> xdot  
time derivative (dimension: nx)
```

Definition at line 80 of file rdata.h.

10.30.5.3 J

```
std::vector<realtyp> J  
Jacobian of differential equation right hand side (dimension: nx x nx, row-major)
```

Definition at line 84 of file rdata.h.

10.30.5.4 z

```
std::vector<realtyp> z  
event output (dimension: nmaxevent x nz, row-major)
```

Definition at line 87 of file rdata.h.

10.30.5.5 sigmaz

```
std::vector<realtyp> sigmaz  
event output sigma standard deviation (dimension: nmaxevent x nz, row-major)  
Definition at line 91 of file rdata.h.
```

10.30.5.6 sz

```
std::vector<realtyp> sz
```

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

Definition at line 95 of file rdata.h.

10.30.5.7 ssigmaz

```
std::vector<realtyp> ssigmaz
```

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

Definition at line 99 of file rdata.h.

10.30.5.8 rz

```
std::vector<realtyp> rz
```

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

Definition at line 102 of file rdata.h.

10.30.5.9 srz

```
std::vector<realtyp> srz
```

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

Definition at line 106 of file rdata.h.

10.30.5.10 s2rz

```
std::vector<realtyp> s2rz
```

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

Definition at line 110 of file rdata.h.

10.30.5.11 x

```
std::vector<realtyp> x  
  
state (dimension: nt x nx, row-major)
```

Definition at line 113 of file rdata.h.

10.30.5.12 sx

```
std::vector<realtyp> sx  
  
parameter derivative of state (dimension: nt x nplist x nx, row-major)
```

Definition at line 117 of file rdata.h.

10.30.5.13 y

```
std::vector<realtyp> y  
  
observable (dimension: nt x ny, row-major)
```

Definition at line 120 of file rdata.h.

10.30.5.14 sigmay

```
std::vector<realtyp> sigmay  
  
observable standard deviation (dimension: nt x ny, row-major)
```

Definition at line 123 of file rdata.h.

10.30.5.15 sy

```
std::vector<realtyp> sy  
  
parameter derivative of observable (dimension: nt x nplist x ny, row-major)
```

Definition at line 127 of file rdata.h.

10.30.5.16 ssigmay

```
std::vector<realtyp> ssigmay
```

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

Definition at line 131 of file rdata.h.

10.30.5.17 res

```
std::vector<realtyp> res
```

observable (dimension: nt*ny, row-major)

Definition at line 134 of file rdata.h.

10.30.5.18 sres

```
std::vector<realtyp> sres
```

parameter derivative of residual (dimension: nt*ny x plist, row-major)

Definition at line 138 of file rdata.h.

10.30.5.19 FIM

```
std::vector<realtyp> FIM
```

fisher information matrix (dimension: plist x plist, row-major)

Definition at line 142 of file rdata.h.

10.30.5.20 numsteps

```
std::vector<int> numsteps
```

number of integration steps forward problem (dimension: nt)

Definition at line 145 of file rdata.h.

10.30.5.21 numstepsB

```
std::vector<int> numstepsB
```

number of integration steps backward problem (dimension: nt)

Definition at line 148 of file rdata.h.

10.30.5.22 numrhsevals

```
std::vector<int> numrhsevals
```

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

Definition at line 151 of file rdata.h.

10.30.5.23 numrhsevalsB

```
std::vector<int> numrhsevalsB
```

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

Definition at line 154 of file rdata.h.

10.30.5.24 numerptestfails

```
std::vector<int> numerptestfails
```

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

Definition at line 157 of file rdata.h.

10.30.5.25 numerptestfailsB

```
std::vector<int> numerptestfailsB
```

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

Definition at line 160 of file rdata.h.

10.30.5.26 numnonlinsolvconvfails

```
std::vector<int> numnonlinsolvconvfails
```

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

Definition at line 164 of file rdata.h.

10.30.5.27 numnonlinsolvconvfailsB

```
std::vector<int> numnonlinsolvconvfailsB
```

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

Definition at line 168 of file rdata.h.

10.30.5.28 order

```
std::vector<int> order
```

employed order forward problem (dimension: nt)

Definition at line 171 of file rdata.h.

10.30.5.29 newton_status

```
int newton_status = 0
```

flag indicating success of Newton solver

Definition at line 174 of file rdata.h.

10.30.5.30 newton_cpu_time

```
double newton_cpu_time = 0.0
```

computation time of the Newton solver [s]

Definition at line 177 of file rdata.h.

10.30.5.31 newton_numsteps

```
std::vector<int> newton_numsteps
```

number of Newton steps for steady state problem (length = 2)

Definition at line 180 of file rdata.h.

10.30.5.32 newton_numlinsteps

```
std::vector<int> newton_numlinsteps
```

number of linear steps by Newton step for steady state problem (length = newton_maxsteps * 2)

Definition at line 183 of file rdata.h.

10.30.5.33 t_steadystate

```
realtype t_steadystate = NAN
```

time at which steadystate was reached in the simulation based approach

Definition at line 186 of file rdata.h.

10.30.5.34 wrms_steadystate

```
realtype wrms_steadystate = NAN
```

weighted root-mean-square of the rhs when steadystate was reached

Definition at line 190 of file rdata.h.

10.30.5.35 wrms_sensi_steadystate

```
realtype wrms_sensi_steadystate = NAN
```

weighted root-mean-square of the rhs when steadystate was reached

Definition at line 194 of file rdata.h.

10.30.5.36 x0

```
std::vector<realtyp> x0
```

preequilibration steady state found by Newton solver (dimension: nx)

Definition at line 197 of file rdata.h.

10.30.5.37 sx0

```
std::vector<realtyp> sx0
```

preequilibration sensitivities found by Newton solver (dimension: plist x nx, row-major)

Definition at line 200 of file rdata.h.

10.30.5.38 llh

```
realtyp llh = 0.0
```

loglikelihood value

Definition at line 203 of file rdata.h.

10.30.5.39 chi2

```
realtyp chi2 = 0.0
```

chi2 value

Definition at line 206 of file rdata.h.

10.30.5.40 sllh

```
std::vector<realtyp> sllh
```

parameter derivative of loglikelihood (dimension: plist)

Definition at line 209 of file rdata.h.

10.30.5.41 s2llh

```
std::vector<realtype> s2llh
```

second order parameter derivative of loglikelihood (dimension: (nJ-1) x plist, row-major)

Definition at line 213 of file rdata.h.

10.30.5.42 status

```
int status = 0
```

status code

Definition at line 216 of file rdata.h.

10.30.5.43 np

```
const int np
```

total number of model parameters

Definition at line 219 of file rdata.h.

10.30.5.44 nk

```
const int nk
```

number of fixed parameters

Definition at line 221 of file rdata.h.

10.30.5.45 nx

```
const int nx
```

number of states

Definition at line 223 of file rdata.h.

10.30.5.46 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 225 of file rdata.h.

10.30.5.47 ny

```
const int ny
```

number of observables

Definition at line 227 of file rdata.h.

10.30.5.48 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 229 of file rdata.h.

10.30.5.49 nz

```
const int nz
```

number of event outputs

Definition at line 231 of file rdata.h.

10.30.5.50 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 233 of file rdata.h.

10.30.5.51 ne

```
const int ne
```

number of events

Definition at line 235 of file rdata.h.

10.30.5.52 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 237 of file rdata.h.

10.30.5.53 nplist

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 240 of file rdata.h.

10.30.5.54 nmaxevent

```
const int nmaxevent
```

maximal number of occurring events (for every event type)

Definition at line 242 of file rdata.h.

10.30.5.55 nt

```
const int nt
```

number of considered timepoints

Definition at line 244 of file rdata.h.

10.30.5.56 newton_maxsteps

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 246 of file rdata.h.

10.30.5.57 pscale

```
std::vector<ParameterScaling> pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 248 of file rdata.h.

10.30.5.58 o2mode

```
const SecondOrderMode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 250 of file rdata.h.

10.30.5.59 sensi

```
const SensitivityOrder sensi
```

sensitivity order

Definition at line 252 of file rdata.h.

10.30.5.60 sensi_meth

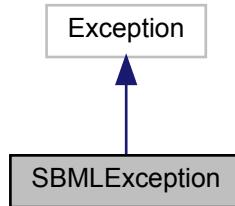
```
const SensitivityMethod sensi_meth
```

sensitivity method

Definition at line 254 of file rdata.h.

10.31 SBMLError Class Reference

Inheritance diagram for SBMLError:



10.31.1 Detailed Description

Definition at line 13 of file sbml_import.py.

10.32 SbmlImporter Class Reference

The [SbmlImporter](#) class generates AMICI C++ files for a model provided in the Systems Biology Markup Language (SBML).

Public Member Functions

- def [`__init__`](#) (self, SBMLFile, `check_validity`=True)
Create a new Model instance.
- def [`reset_symbols`](#) (self)
Reset the symbols attribute to default values.
- def [`loadSBMLFile`](#) (self, SBMLFile)
Parse the provided SBML file.
- def [`checkLibSBMLErrors`](#) (self)
Checks the error log in the current self.sbml_doc.
- def [`sbml2amici`](#) (self, modelName, output_dir=None, observables=None, constantParameters=None, sigmas=None, verbose=False, assume_pow_positivity=False, compiler=None)
Generate AMICI C++ files for the model provided to the constructor.
- def [`processSBML`](#) (self, constantParameters=None)
Read parameters, species, reactions, and so on from SBML model.
- def [`checkSupport`](#) (self)
Check whether all required SBML features are supported.
- def [`processSpecies`](#) (self)
Get species information from SBML model.
- def [`processParameters`](#) (self, constantParameters=None)
Get parameter information from SBML model.
- def [`processCompartments`](#) (self)

- def `processReactions` (self)

Get compartment information, stoichiometric matrix and fluxes from SBML model.
- def `processRules` (self)

Process Rules defined in the SBML model.
- def `processVolumeConversion` (self)

Convert equations from amount to volume.
- def `processTime` (self)

Convert time_symbol into cpp variable.
- def `processObservables` (self, observables, sigmas)

Perform symbolic computations required for objective function evaluation.
- def `replaceInAllExpressions` (self, old, new)

Replace 'old' by 'new' in all symbolic expressions.
- def `cleanReservedSymbols` (self)

Remove all reserved symbols from self.symbols.
- def `replaceSpecialConstants` (self)

Replace all special constants by their respective SBML csymbol definition.

Public Attributes

- `check_validity`

indicates whether the validity of the SBML document
- `symbols`

dict carrying symbolic definitions
Type: dict
- `SBMLReader`

the libSBML sbml reader [!not storing this will result
- `sbml_doc`

document carrying the sbml defintion [!not storing this
- `sbml`

sbml definition [!not storing this will result in a segfault!]
- `speciesIndex`

maps species names to indices
Type: dict
- `speciesCompartments`

compartment for each species
Type:
- `constantSpecies`

ids of species that are marked as constant
Type: list
- `boundaryConditionSpecies`

ids of species that are marked as boundary
- `speciesHasOnlySubstanceUnits`

flags indicating whether a species has
- `speciesConversionFactor`

conversion factors for every species
Type:
- `compartmentSymbols`

compartment ids
Type: sympy.Matrix
- `compartmentVolume`

numeric/symbolic compartment volumes

Type:

- **stoichiometricMatrix**

stoichiometry matrix of the model

Type:

- **fluxVector**

reaction kinetic laws

Type: sympy.Matrix

10.32.1 Detailed Description

Definition at line 35 of file sbml_import.py.

10.32.2 Constructor & Destructor Documentation

10.32.2.1 __init__()

```
def __init__ (
    self,
    SBMLFile,
    check_validity = True )
```

Parameters

SBMLFile	Path to SBML file where the model is specified Type: string
check_validity	Flag indicating whether the validity of the SBML document should be checked Type: bool

Returns

[SbmlImporter](#) instance with attached SBML document

Definition at line 143 of file sbml_import.py.

10.32.3 Member Function Documentation

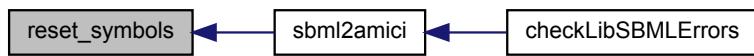
10.32.3.1 reset_symbols()

```
def reset_symbols (
    self )
```

Returns

Definition at line 162 of file sbml_import.py.

Here is the caller graph for this function:

**10.32.3.2 loadSBMLFile()**

```
def loadSBMLFile (
    self,
    SBMLFile )
```

Parameters

<i>SBMLFile</i>	path to SBML file
	Type: str

Returns

Definition at line 175 of file sbml_import.py.

10.32.3.3 checkLibSBMLErrors()

```
def checkLibSBMLErrors (
    self )
```

Returns

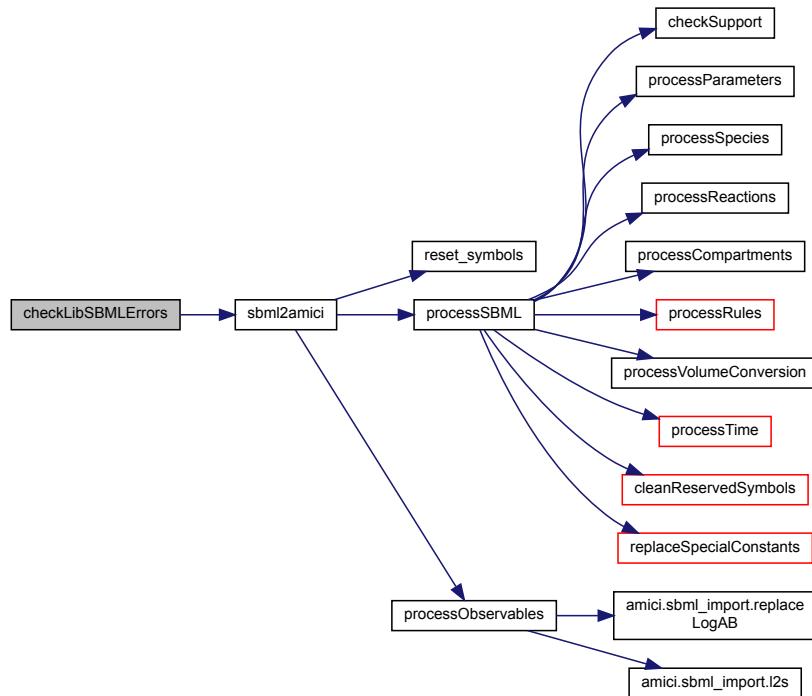
raises SBMLError if errors with severity ERROR or FATAL have

Exceptions

<i>occured</i>	
----------------	--

Definition at line 216 of file sbml_import.py.

Here is the call graph for this function:



10.32.3.4 sbml2amici()

```

def sbml2amici (
    self,
    modelName,
    output_dir = None,
    observables = None,
    constantParameters = None,
    sigmas = None,
    verbose = False,
    assume_pow_positivity = False,
    compiler = None )

```

Parameters

<i>modelName</i>	name of the model/model directory Type: str
<i>output_dir</i>	see <code>sbml_import.setPaths()</code> Type: str
<i>observables</i>	dictionary(observableId:{'name':observableName (optional), 'formula':formulaString}) to be added to the model Type: dict
<i>sigmas</i>	dictionary(observableId: sigma value or (existing) parameter name) Type: dict

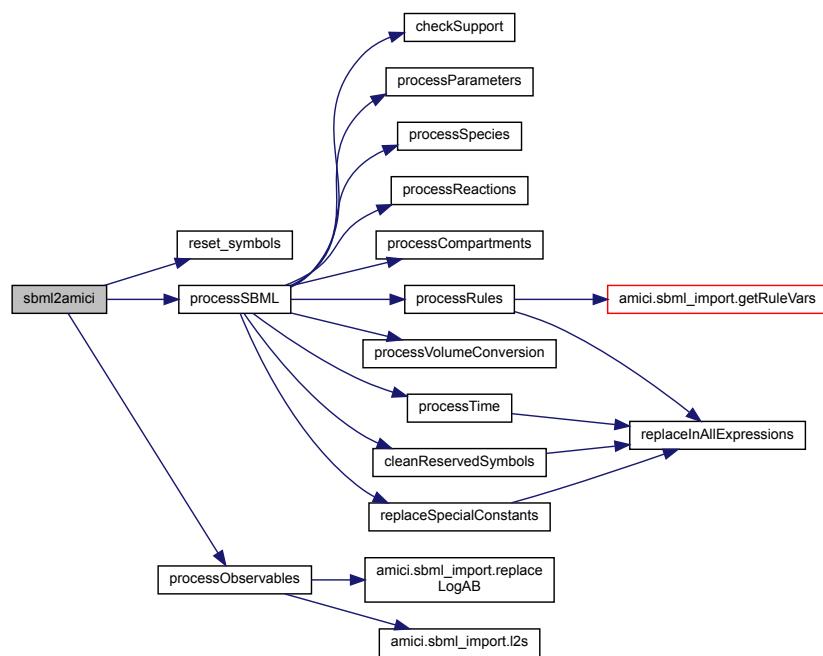
Parameters

<i>constantParameters</i>	list of SBML Ids identifying constant parameters Type: dict
<i>verbose</i>	more verbose output if True Type: bool
<i>assume_pow_positivity</i>	if set to true, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors Type: bool
<i>compiler</i>	distutils/setuptools compiler selection to build the python extension Type: str

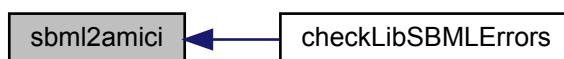
Returns

Definition at line 274 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.32.3.5 processSBML()

```
def processSBML (
    self,
    constantParameters = None )
```

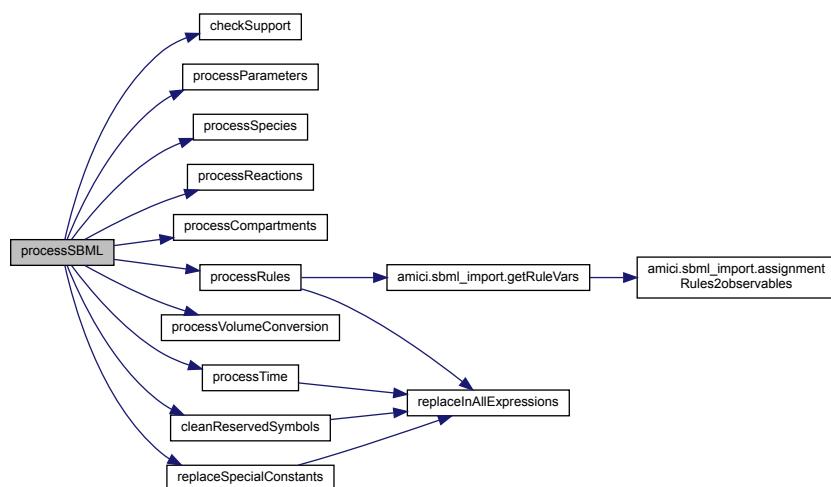
Parameters

<i>constantParameters</i>	SBML Ids identifying constant parameters Type: list
---------------------------	---

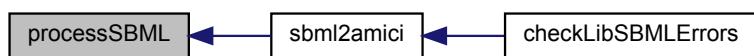
Returns

Definition at line 312 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.32.3.6 checkSupport()

```
def checkSupport ( self )
```

Returns

Definition at line 337 of file sbml_import.py.

Here is the caller graph for this function:



10.32.3.7 processSpecies()

```
def processSpecies ( self )
```

Returns

Definition at line 374 of file sbml_import.py.

Here is the caller graph for this function:



10.32.3.8 processParameters()

```
def processParameters ( self, constantParameters = None )
```

Parameters

<i>constantParameters</i>	SBML Ids identifying constant parameters Type: list
---------------------------	---

Returns

Definition at line 466 of file sbml_import.py.

Here is the caller graph for this function:

**10.32.3.9 processCompartments()**

```
def processCompartments (
    self )
```

Returns

Definition at line 543 of file sbml_import.py.

Here is the caller graph for this function:



10.32.3.10 processReactions()

```
def processReactions (
    self )
```

Returns

Definition at line 578 of file sbml_import.py.

Here is the caller graph for this function:



10.32.3.11 processRules()

```
def processRules (
    self )
```

Returns

Definition at line 690 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.32.3.12 processVolumeConversion()

```
def processVolumeConversion (
    self )
```

Returns

Definition at line 767 of file sbml_import.py.

Here is the caller graph for this function:



10.32.3.13 processTime()

```
def processTime (
    self )
```

Returns

Definition at line 789 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.32.3.14 processObservables()

```
def processObservables (
    self,
    observables,
    sigmas )
```

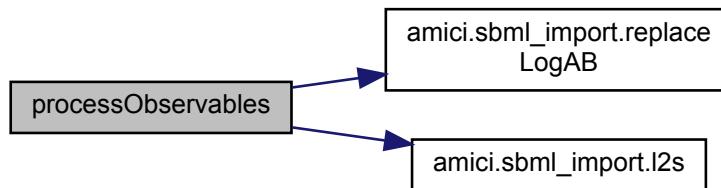
Parameters

<i>observables</i>	dictionary(observableId:{'name':observableName (optional), 'formula':formulaString}) to be added to the model Type: dict
<i>sigmas</i>	dictionary(observableId: sigma value or (existing) parameter name) Type: dict

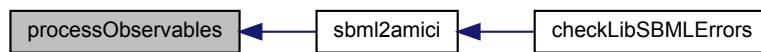
Returns

Definition at line 811 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.32.3.15 replaceInAllExpressions()**

```
def replaceInAllExpressions (
    self,
    old,
    new )
```

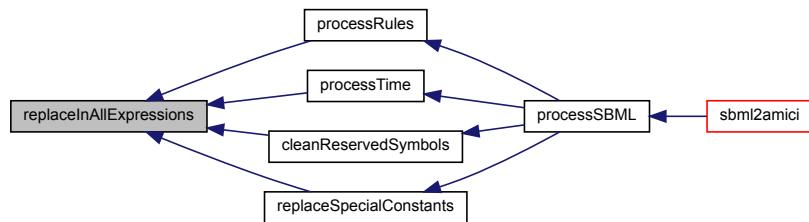
Parameters

<i>old</i>	symbolic variables to be replaced Type: symengine.Symbol
<i>new</i>	replacement symbolic variables Type: symengine.Symbol

Returns

Definition at line 918 of file sbml_import.py.

Here is the caller graph for this function:

**10.32.3.16 cleanReservedSymbols()**

```
def cleanReservedSymbols ( self )
```

Returns

Definition at line 945 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



10.32.3.17 replaceSpecialConstants()

```
def replaceSpecialConstants (
    self )
```

Returns

Definition at line 966 of file sbml_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:

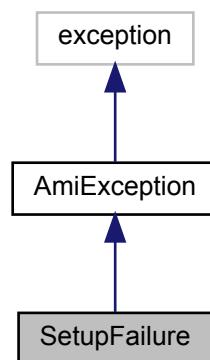


10.33 SetupFailure Class Reference

setup failure exception this exception should be thrown when the solver setup failed for this exception we can assume that we cannot recover from the exception and an error will be thrown

```
#include <exception.h>
```

Inheritance diagram for SetupFailure:



Public Member Functions

- [SetupFailure](#) (const char *msg)

10.33.1 Detailed Description

Definition at line 188 of file exception.h.

10.33.2 Constructor & Destructor Documentation

10.33.2.1 SetupFailure()

```
SetupFailure (
    const char * msg )
```

constructor, simply calls [AmiException](#) constructor

Parameters

<i>msg</i>	<input type="text"/>
------------	----------------------

Definition at line 193 of file exception.h.

10.34 Solver Class Reference

```
#include <solver.h>
```

Public Member Functions

- [Solver](#) (const [Solver](#) &other)

Solver copy constructor.
- virtual [Solver](#) * [clone](#) () const =0

Clone this instance.
- void [setup](#) ([AmiVector](#) *x, [AmiVector](#) *dx, [AmiVectorArray](#) *sx, [AmiVectorArray](#) *sdx, [Model](#) *model)

Initialises the ami memory object and applies specified options.
- void [setupAMIB](#) ([BackwardProblem](#) *bwd, [Model](#) *model)
- virtual void [getSens](#) ([realtype](#) *tret, [AmiVectorArray](#) *yySout) const =0
- void [getDiagnosis](#) (const int it, [ReturnData](#) *rdata) const
- void [getDiagnosisB](#) (const int it, [ReturnData](#) *rdata, int which) const
- virtual void [getRootInfo](#) (int *rootsfound) const =0
- virtual void [reInit](#) ([realtype](#) t0, [AmiVector](#) *yy0, [AmiVector](#) *yp0)=0
- virtual void [sensReInit](#) ([AmiVectorArray](#) *yS0, [AmiVectorArray](#) *ypS0)=0
- virtual void [calcIC](#) ([realtype](#) tout1, [AmiVector](#) *x, [AmiVector](#) *dx)=0
- virtual void [calcICB](#) (int which, [realtype](#) tout1, [AmiVector](#) *xB, [AmiVector](#) *dxB)=0
- virtual int [solve](#) ([realtype](#) tout, [AmiVector](#) *yret, [AmiVector](#) *ypret, [realtype](#) *tret, int itask)=0
- virtual int [solveF](#) ([realtype](#) tout, [AmiVector](#) *yret, [AmiVector](#) *ypret, [realtype](#) *tret, int itask, int *ncheckPtr)=0

- virtual void `solveB` (realtype tBout, int itaskB)=0
- virtual void `setStopTime` (realtype tstop)=0
- virtual void `reInitB` (int which, realtype tB0, AmiVector *yyB0, AmiVector *ypB0)=0
- virtual void `getB` (int which, realtype *tret, AmiVector *yy, AmiVector *yp) const =0
- virtual void `getQuadB` (int which, realtype *tret, AmiVector *qB) const =0
- virtual void `quadReInitB` (int which, AmiVector *yQB0)=0
- virtual void `turnOffRootFinding` ()=0
- `SensitivityMethod getSensitivityMethod () const`
- void `setSensitivityMethod` (`SensitivityMethod` sensi_meth)
 `setSensitivityMethod`
- int `getNewtonMaxSteps () const`
 `getNewtonMaxSteps`
- void `setNewtonMaxSteps` (int newton_maxsteps)
 `setNewtonMaxSteps`
- bool `getNewtonPreequilibration () const`
 `getNewtonPreequilibration`
- void `setNewtonPreequilibration` (bool newton_preq)
 `setNewtonPreequilibration`
- int `getNewtonMaxLinearSteps () const`
 `getNewtonMaxLinearSteps`
- void `setNewtonMaxLinearSteps` (int newton_maxlinsteps)
 `setNewtonMaxLinearSteps`
- `SensitivityOrder getSensitivityOrder () const`
 returns the sensitivity order
- void `setSensitivityOrder` (`SensitivityOrder` sensi)
 sets the sensitivity order
- double `getRelativeTolerance () const`
 returns the relative tolerances for the forward & backward problem
- void `setRelativeTolerance` (double rtol)
 sets the relative tolerances for the forward & backward problem
- double `getAbsoluteTolerance () const`
 returns the absolute tolerances for the forward & backward problem
- void `setAbsoluteTolerance` (double atol)
 sets the absolute tolerances for the forward & backward problem
- double `getRelativeToleranceSensi () const`
 returns the relative tolerances for the forward sensitivity problem
- void `setRelativeToleranceSensi` (double rtol)
 sets the relative tolerances for the forward sensitivity problem
- double `getAbsoluteToleranceSensi () const`
 returns the absolute tolerances for the forward sensitivity problem
- void `setAbsoluteToleranceSensi` (double atol)
 sets the absolute tolerances for the forward sensitivity problem
- double `getRelativeToleranceQuadratures () const`
 returns the relative tolerance for the quadrature problem
- void `setRelativeToleranceQuadratures` (double rtol)
 sets the relative tolerance for the quadrature problem
- double `getAbsoluteToleranceQuadratures () const`
 returns the absolute tolerance for the quadrature problem
- void `setAbsoluteToleranceQuadratures` (double atol)
 sets the absolute tolerance for the quadrature problem
- double `getRelativeToleranceSteadyState () const`

- `void setRelativeToleranceSteadyState (double rtol)`
sets the relative tolerance for the steady state problem
- `double getAbsoluteToleranceSteadyState () const`
returns the absolute tolerance for the steady state problem
- `void setAbsoluteToleranceSteadyState (double atol)`
sets the absolute tolerance for the steady state problem
- `double getRelativeToleranceSteadyStateSensi () const`
returns the relative tolerance for the sensitivities of the steady state problem
- `void setRelativeToleranceSteadyStateSensi (double rtol)`
sets the relative tolerance for the sensitivities of the steady state problem
- `double getAbsoluteToleranceSteadyStateSensi () const`
returns the absolute tolerance for the sensitivities of the steady state problem
- `int getMaxSteps () const`
returns the maximum number of solver steps for the forward problem
- `void setMaxSteps (int maxsteps)`
sets the maximum number of solver steps for the forward problem
- `int getMaxStepsBackwardProblem () const`
returns the maximum number of solver steps for the backward problem
- `void setMaxStepsBackwardProblem (int maxsteps)`
sets the maximum number of solver steps for the backward problem
- `LinearMultistepMethod getLinearMultistepMethod () const`
returns the linear system multistep method
- `void setLinearMultistepMethod (LinearMultistepMethod lmm)`
sets the linear system multistep method
- `NonlinearSolverIteration getNonlinearSolverIteration () const`
returns the nonlinear system solution method
- `void setNonlinearSolverIteration (NonlinearSolverIteration iter)`
sets the nonlinear system solution method
- `InterpolationType getInterpolationType () const`
`getInterpolationType`
- `void setInterpolationType (InterpolationType interpType)`
sets the interpolation of the forward solution that is used for the backwards problem
- `StateOrdering getStateOrdering () const`
sets KLU state ordering mode
- `void setStateOrdering (StateOrdering ordering)`
sets KLU state ordering mode (only applies when linsol is set to amici.AMICI_KLU)
- `booleantype getStabilityLimitFlag () const`
returns stability limit detection mode
- `void setStabilityLimitFlag (booleantype stldet)`
set stability limit detection mode
- `LinearSolver getLinearSolver () const`
`getLinearSolver`
- `void setLinearSolver (LinearSolver linsol)`
`setLinearSolver`
- `InternalSensitivityMethod getInternalSensitivityMethod () const`
returns the internal sensitivity method
- `void setInternalSensitivityMethod (InternalSensitivityMethod ism)`
sets the internal sensitivity method

Protected Member Functions

- virtual void `init (AmiVector *x, AmiVector *dx, realtype t)=0`
- virtual void `binit (int which, AmiVector *xB, AmiVector *dxB, realtype t)=0`
- virtual void `qbinit (int which, AmiVector *qBdot)=0`
- virtual void `rootInit (int ne)=0`
- virtual void `sensInit1 (AmiVectorArray *sx, AmiVectorArray *sdx, int nplist)=0`
- virtual void `setDenseJacFn ()=0`
- virtual void `setSparseJacFn ()=0`
- virtual void `setBandJacFn ()=0`
- virtual void `setJacTimesVecFn ()=0`
- virtual void `setDenseJacFnB (int which)=0`
- virtual void `setSparseJacFnB (int which)=0`
- virtual void `setBandJacFnB (int which)=0`
- virtual void `setJacTimesVecFnB (int which)=0`
- virtual void `allocateSolver ()=0`
- virtual void `setSStolerances (double rtol, double atol)=0`
- virtual void `setSensSStolerances (double rtol, double *atol)=0`
- virtual void `setSensErrCon (bool error_corr)=0`
- virtual void `setQuadErrConB (int which, bool flag)=0`
- virtual void `setErrorHandlerFn ()=0`
- virtual void `setUserData (Model *model)=0`
- virtual void `setUserDataB (int which, Model *model)=0`
- virtual void `setMaxNumSteps (long int mxsteps)=0`
- virtual void `setMaxNumStepsB (int which, long int mxstepsB)=0`
- virtual void `setStabLimDet (int stldet)=0`
- virtual void `setStabLimDetB (int which, int stldet)=0`
- virtual void `setId (Model *model)=0`
- virtual void `setSuppressAlg (bool flag)=0`
- virtual void `setSensParams (realtype *p, realtype *pbar, int *plist)=0`
- virtual void `getDky (realtype t, int k, AmiVector *dky) const =0`
- virtual void `adjInit ()=0`
- virtual void `allocateSolverB (int *which)=0`
- virtual void `setSStolerancesB (int which, realtype relTolB, realtype absTolB)=0`
- virtual void `quadSStolerancesB (int which, realtype reltolQB, realtype abstolQB)=0`
- virtual void `dense (int nx)=0`
- virtual void `denseB (int which, int nx)=0`
- virtual void `band (int nx, int ubw, int lbw)=0`
- virtual void `bandB (int which, int nx, int ubw, int lbw)=0`
- virtual void `diag ()=0`
- virtual void `diagB (int which)=0`
- virtual void `spgmr (int prectype, int maxl)=0`
- virtual void `spgmrB (int which, int prectype, int maxl)=0`
- virtual void `spbcg (int prectype, int maxl)=0`
- virtual void `spbcgB (int which, int prectype, int maxl)=0`
- virtual void `sptfqmr (int prectype, int maxl)=0`
- virtual void `sptfqmrB (int which, int prectype, int maxl)=0`
- virtual void `klu (int nx, int nnz, int sparsetype)=0`
- virtual void `kluSetOrdering (int ordering)=0`
- virtual void `kluSetOrderingB (int which, int ordering)=0`
- virtual void `kluB (int which, int nx, int nnz, int sparsetype)=0`
- virtual void `getNumSteps (void *ami_mem, long int *numsteps) const =0`
- virtual void `getNumRhsEvals (void *ami_mem, long int *numrhsvals) const =0`
- virtual void `getNumErrTestFails (void *ami_mem, long int *numerertestfails) const =0`
- virtual void `getNumNonlinSolvConvFails (void *ami_mem, long int *numnonlinconvfails) const =0`

- virtual void `getLastOrder` (void *ami_mem, int *order) const =0
- void `initializeLinearSolver` (const `Model` *model)
- void `initializeLinearSolverB` (const `Model` *model, const int which)
- virtual int `nplist` () const =0
- virtual int `nx` () const =0
- virtual const `Model` * `getModel` () const =0
- virtual bool `getMallocDone` () const =0
- virtual bool `getAdjMallocDone` () const =0
- virtual void * `getAdjBmem` (void *ami_mem, int which)=0
- void `applyTolerances` ()
- void `applyTolerancesFSA` ()
- void `applyTolerancesASA` (int which)
- void `applyQuadTolerancesASA` (int which)
- void `applySensitivityTolerances` ()

Static Protected Member Functions

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

Protected Attributes

- std::unique_ptr< void, std::function< void(void *)> > `solverMemory`
- std::vector< std::unique_ptr< void, std::function< void(void *)> > > > `solverMemoryB`
- bool `solverWasCalled` = false
- `InternalSensitivityMethod` `ism` = `InternalSensitivityMethod`::simultaneous
- `LinearMultistepMethod` `lmm` = `LinearMultistepMethod`::BDF
- `NonlinearSolverIteration` `iter` = `NonlinearSolverIteration`::newton
- `InterpolationType` `interpType` = `InterpolationType`::hermite
- int `maxsteps` = 10000

Friends

- template<class Archive >
void `boost::serialization::serialize` (Archive &ar, `Solver` &r, const unsigned int version)
Serialize Solver (see boost::serialization::serialize)
- bool `operator==` (const `Solver` &a, const `Solver` &b)
Check equality of data members.

10.34.1 Detailed Description

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

Definition at line 38 of file solver.h.

10.34.2 Constructor & Destructor Documentation

10.34.2.1 `Solver()`

```
Solver (
    const Solver & other )
```

Parameters

<i>other</i>	<input type="checkbox"/>
--------------	--------------------------

Definition at line 46 of file solver.h.

10.34.3 Member Function Documentation

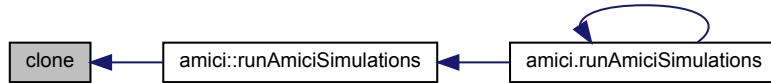
10.34.3.1 clone()

```
virtual Solver* clone ( ) const [pure virtual]
```

Returns

The clone

Here is the caller graph for this function:



10.34.3.2 setup()

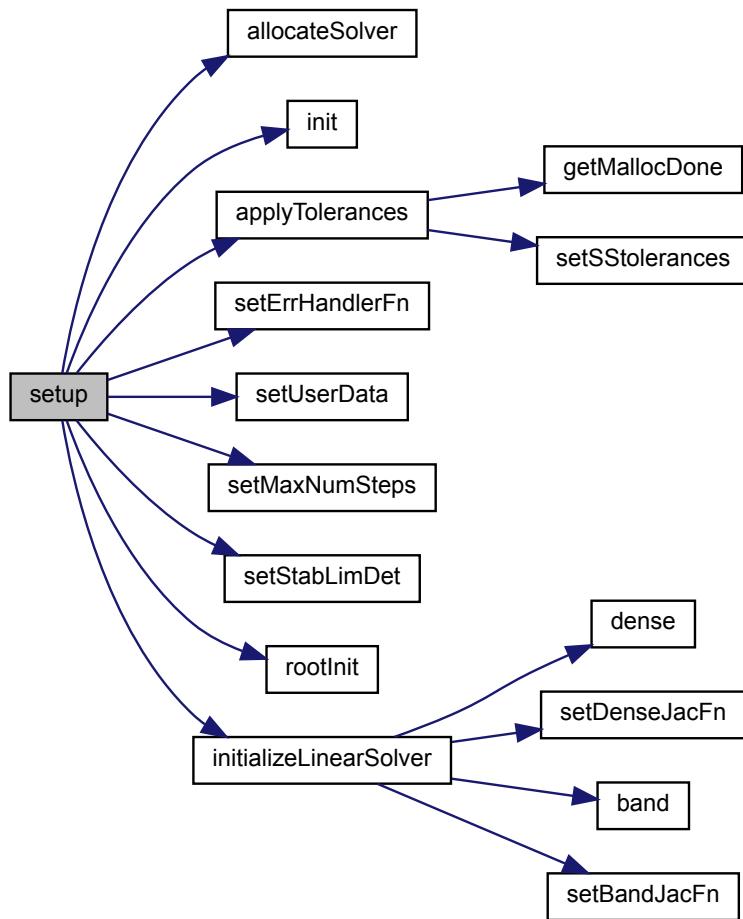
```
void setup (
    AmiVector * x,
    AmiVector * dx,
    AmiVectorArray * sx,
    AmiVectorArray * sdx,
    Model * model )
```

Parameters

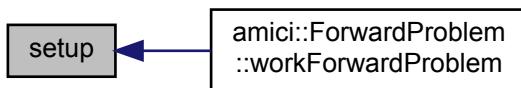
<i>x</i>	state vector
<i>dx</i>	state derivative vector (DAE only)
<i>sx</i>	state sensitivity vector
<i>sdx</i>	state derivative sensitivity vector (DAE only)
<i>model</i>	pointer to the model object

Definition at line 27 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.3 setupAMIB()

```
void setupAMIB (
    BackwardProblem * bwd,
    Model * model )
```

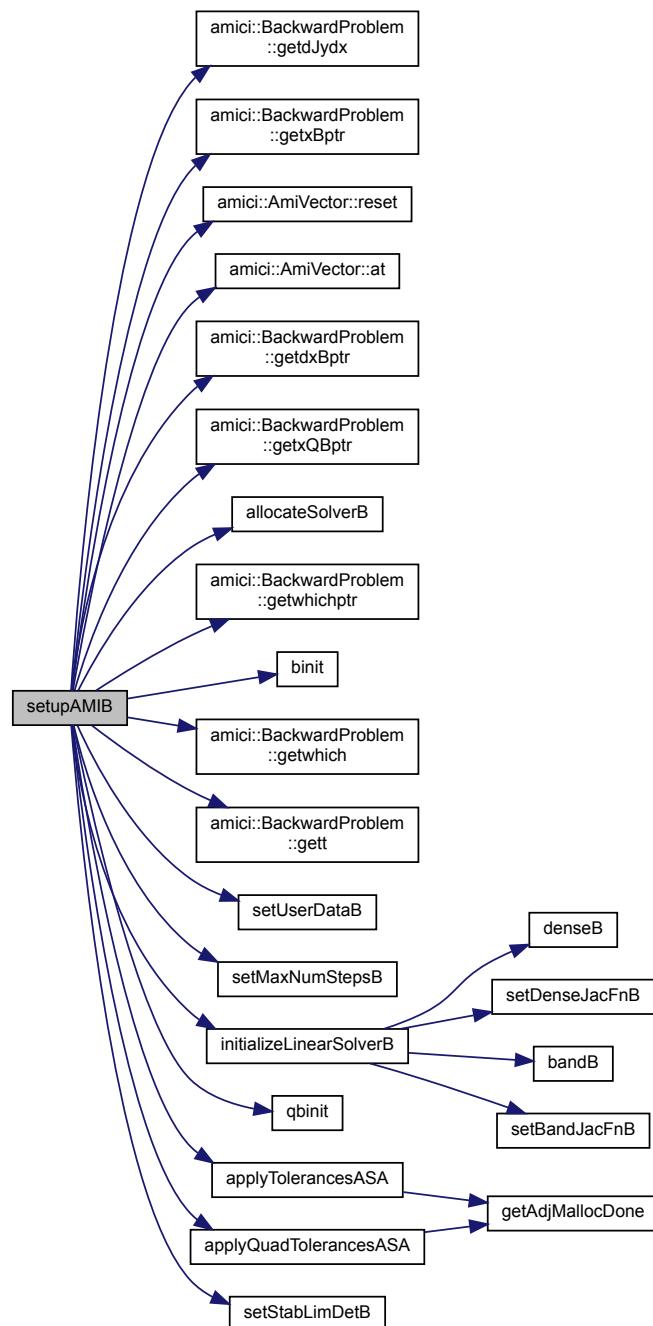
`setupAMIB` initialises the AMI memory object for the backwards problem

Parameters

<code>bwd</code>	pointer to backward problem
<code>model</code>	pointer to the model object

Definition at line 100 of file `solver.cpp`.

Here is the call graph for this function:



10.34.3.4 getSens()

```
virtual void getSens (
    realtype * tret,
    AmiVectorArray * yySout ) const [pure virtual]
```

getSens extracts diagnosis information from solver memory block and writes them into the return data instance

Parameters

<i>tret</i>	time at which the sensitivities should be computed
<i>yySout</i>	vector with sensitivities

10.34.3.5 getDiagnosis()

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

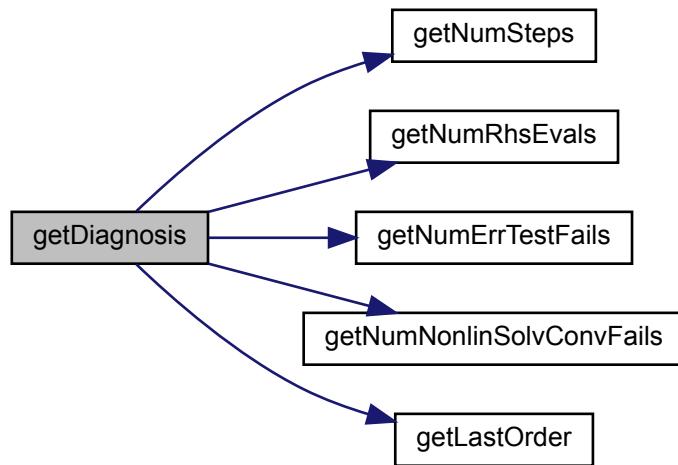
getDiagnosis extracts diagnosis information from solver memory block and writes them into the return data object

Parameters

<i>it</i>	time-point index
<i>rdata</i>	pointer to the return data object

Definition at line 184 of file solver.cpp.

Here is the call graph for this function:



10.34.3.6 `getDiagnosisB()`

```
void getDiagnosisB (
    const int it,
    ReturnData * rdata,
    int which ) const
```

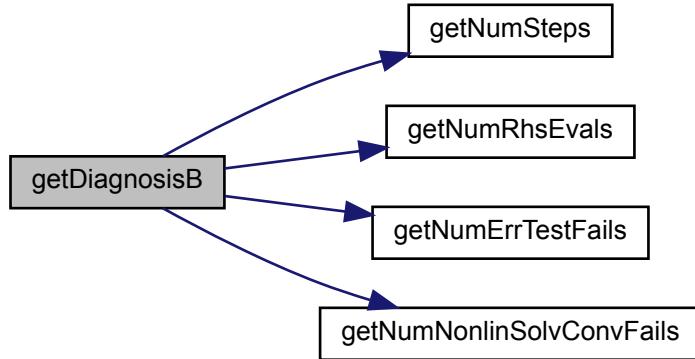
`getDiagnosisB` extracts diagnosis information from solver memory block and writes them into the return data object for the backward problem

Parameters

<i>it</i>	time-point index
<i>rdata</i>	pointer to the return data object
<i>which</i>	identifier of the backwards problem

Definition at line 204 of file `solver.cpp`.

Here is the call graph for this function:



10.34.3.7 getRootInfo()

```
virtual void getRootInfo (
    int * rootsfound ) const [pure virtual]
```

getRootInfo extracts information which event occured

Parameters

<i>rootsfound</i>	array with flags indicating whether the respective event occured
-------------------	--

10.34.3.8 reInit()

```
virtual void reInit (
    realtype t0,
    AmiVector * yy0,
    AmiVector * yp0 ) [pure virtual]
```

ReInit reinitializes the states in the solver after an event occurence

Parameters

<i>t0</i>	new timepoint
<i>yy0</i>	new state variables
<i>yp0</i>	new derivative state variables (DAE only)

10.34.3.9 sensReInit()

```
virtual void sensReInit (
    AmiVectorArray * yS0,
    AmiVectorArray * ypS0 ) [pure virtual]
```

SensReInit reinitializes the state sensitivities in the solver after an event occurrence

Parameters

<i>yS0</i>	new state sensitivity
<i>ypS0</i>	new derivative state sensitivities (DAE only)

10.34.3.10 calcIC()

```
virtual void calcIC (
    realtype tout1,
    AmiVector * x,
    AmiVector * dx ) [pure virtual]
```

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

Parameters

<i>tout1</i>	next timepoint to be computed (sets timescale)
<i>x</i>	initial state variables
<i>dx</i>	initial derivative state variables (DAE only)

10.34.3.11 calcICB()

```
virtual void calcICB (
    int which,
    realtype tout1,
    AmiVector * xB,
    AmiVector * dxB ) [pure virtual]
```

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

Parameters

<i>which</i>	identifier of the backwards problem
<i>tout1</i>	next timepoint to be computed (sets timescale)
<i>xB</i>	states of final solution of the forward problem
<i>dxB</i>	derivative states of final solution of the forward problem (DAE only)

10.34.3.12 solve()

```
virtual int solve (
    realtype tout,
    AmiVector * yret,
    AmiVector * ypre,
    realtype * tret,
    int itask ) [pure virtual]
```

Solve solves the forward problem until a predefined timepoint

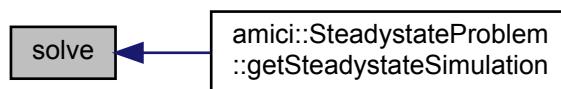
Parameters

<i>tout</i>	timepoint until which simulation should be performed
<i>yret</i>	states
<i>ypre</i>	derivative states (DAE only)
<i>tret</i>	pointer to the time variable
<i>itask</i>	task identifier, can be CV_NORMAL or CV_ONE_STEP

Returns

status flag indicating success of execution

Here is the caller graph for this function:



10.34.3.13 solveF()

```
virtual int solveF (
    realtype tout,
    AmiVector * yret,
    AmiVector * ypre,
    realtype * tret,
    int itask,
    int * ncheckPtr ) [pure virtual]
```

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

Parameters

<i>tout</i>	timepoint until which simulation should be performed
<i>yret</i>	states
<i>ypre</i>	derivative states (DAE only)
<i>tret</i>	pointer to the time variable
<i>itask</i>	task identifier, can be CV_NORMAL or CV_ONE_STEP
<i>ncheckPtr</i>	pointer to a number that counts the internal checkpoints

Returns

status flag indicating success of execution

10.34.3.14 solveB()

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

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

Parameters

<i>tBout</i>	timepoint until which simulation should be performed
<i>itaskB</i>	task identifier, can be CV_NORMAL or CV_ONE_STEP

10.34.3.15 setStopTime()

```
virtual void setStopTime (
    realtype tstop ) [pure virtual]
```

SetStopTime sets a timepoint at which the simulation will be stopped

Parameters

<i>tstop</i>	timepoint until which simulation should be performed
--------------	--

10.34.3.16 reInitB()

```
virtual void reInitB (
    int which,
    realtype tB0,
    AmiVector * yyB0,
    AmiVector * ypB0 ) [pure virtual]
```

ReInitB reinitializes the adjoint states after an event occurence

Parameters

<i>which</i>	identifier of the backwards problem
<i>tB0</i>	new timepoint
<i>yyB0</i>	new adjoint state variables
<i>ypB0</i>	new adjoint derivative state variables (DAE only)

10.34.3.17 getB()

```
virtual void getB (
    int which,
    realtype * tret,
    AmiVector * yy,
    AmiVector * yp ) const [pure virtual]
```

getB returns the current adjoint states

Parameters

<i>which</i>	identifier of the backwards problem
<i>tret</i>	time at which the adjoint states should be computed
<i>yy</i>	adjoint state variables
<i>yp</i>	adjoint derivative state variables (DAE only)

10.34.3.18 getQuadB()

```
virtual void getQuadB (
    int which,
    realtype * tret,
    AmiVector * qB ) const [pure virtual]
```

getQuadB returns the current adjoint states

Parameters

<i>which</i>	identifier of the backwards problem
<i>tret</i>	time at which the adjoint states should be computed
<i>qB</i>	adjoint quadrature state variables

10.34.3.19 quadReInitB()

```
virtual void quadReInitB (
    int which,
    AmiVector * yQBO ) [pure virtual]
```

ReInitB reinitializes the adjoint states after an event occurrence

Parameters

<i>which</i>	identifier of the backwards problem
<i>yQBO</i>	new adjoint quadrature state variables

10.34.3.20 turnOffRootFinding()

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

turnOffRootFinding disables rootfinding

10.34.3.21 getSensitivityMethod()

```
SensitivityMethod getSensitivityMethod ( ) const
```

sensitivity method

Returns

method enum

Definition at line 471 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.22 setSensitivityMethod()**

```
void setSensitivityMethod (
    SensitivityMethod sensi_meth )
```

Parameters

<i>sensi_meth</i>	<input type="text"/>
-------------------	----------------------

Definition at line 475 of file solver.cpp.

Here is the caller graph for this function:



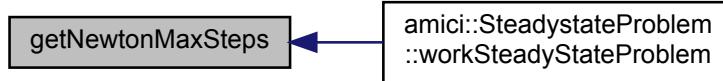
10.34.3.23 getNewtonMaxSteps()

```
int getNewtonMaxSteps ( ) const
```

Returns

Definition at line 479 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.24 setNewtonMaxSteps()

```
void setNewtonMaxSteps ( int newton_maxsteps )
```

Parameters

<i>newton_maxsteps</i>	
------------------------	--

Definition at line 483 of file solver.cpp.

Here is the caller graph for this function:



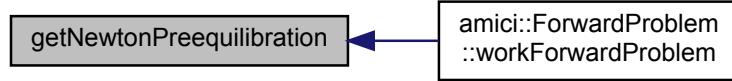
10.34.3.25 getNewtonPreequilibration()

```
bool getNewtonPreequilibration ( ) const
```

Returns

Definition at line 489 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.26 setNewtonPreequilibration()

```
void setNewtonPreequilibration ( bool newton_preeq )
```

Parameters

newton_preeq	
--------------	--

Definition at line 493 of file solver.cpp.

Here is the caller graph for this function:



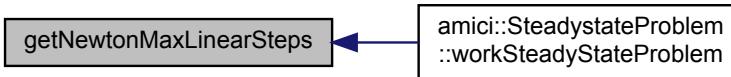
10.34.3.27 getNewtonMaxLinearSteps()

```
int getNewtonMaxLinearSteps ( ) const
```

Returns

Definition at line 497 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.28 setNewtonMaxLinearSteps()

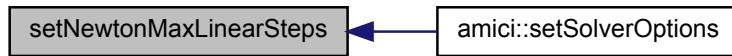
```
void setNewtonMaxLinearSteps ( int newton_maxlinsteps )
```

Parameters

<i>newton_maxlinsteps</i>

Definition at line 501 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.29 getSensitivityOrder()

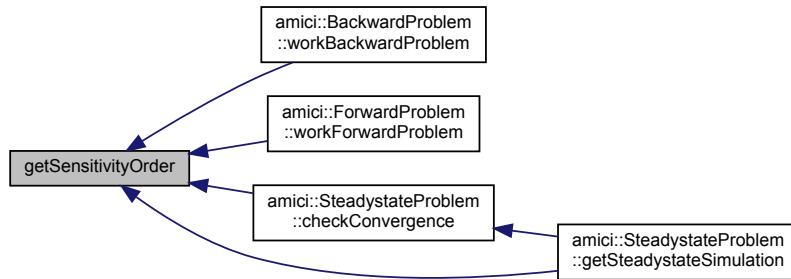
`SensitivityOrder getSensitivityOrder () const`

Returns

sensitivity order

Definition at line 507 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.30 setSensitivityOrder()

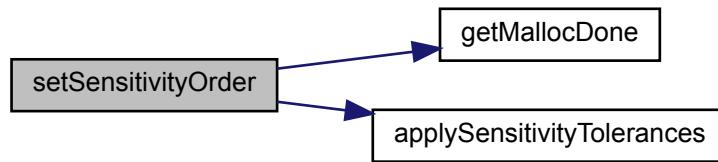
```
void setSensitivityOrder (
    SensitivityOrder sensi )
```

Parameters

<code>sensi</code>	sensitivity order
--------------------	-------------------

Definition at line 511 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.31 `getRelativeTolerance()`

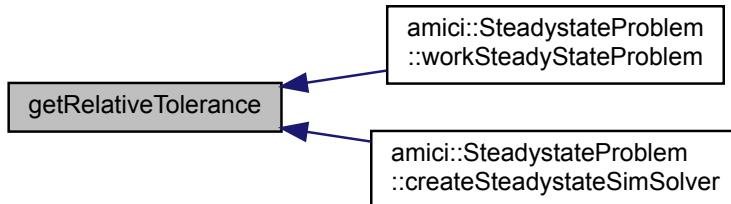
```
double getRelativeTolerance ( ) const
```

Returns

relative tolerances

Definition at line 518 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.32 setRelativeTolerance()

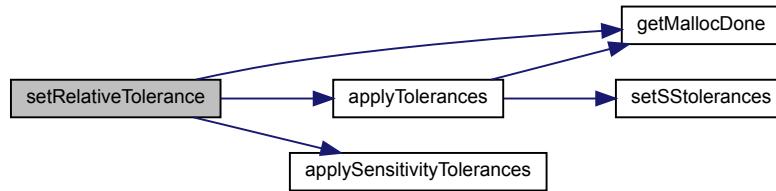
```
void setRelativeTolerance (
    double rtol )
```

Parameters

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 522 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.34.3.33 getAbsoluteTolerance()**

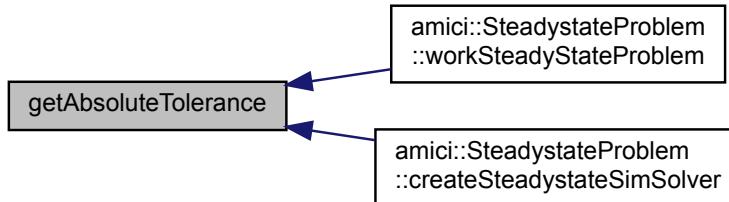
```
double getAbsoluteTolerance ( ) const
```

Returns

absolute tolerances

Definition at line 534 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.34 setAbsoluteTolerance()

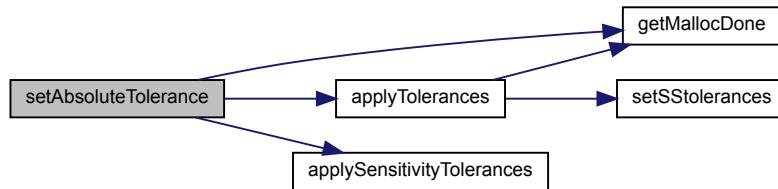
```
void setAbsoluteTolerance (
    double atol )
```

Parameters

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 538 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.35 getRelativeToleranceSensi()

```
double getRelativeToleranceSensi ( ) const
```

Returns

relative tolerances

Definition at line 550 of file solver.cpp.

Here is the call graph for this function:



10.34.3.36 setRelativeToleranceSensi()

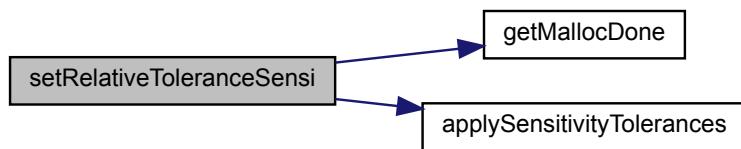
```
void setRelativeToleranceSensi (
    double rtol )
```

Parameters

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 554 of file solver.cpp.

Here is the call graph for this function:



10.34.3.37 getAbsoluteToleranceSensi()

```
double getAbsoluteToleranceSensi ( ) const
```

Returns

absolute tolerances

Definition at line 565 of file solver.cpp.

Here is the call graph for this function:



10.34.3.38 setAbsoluteToleranceSensi()

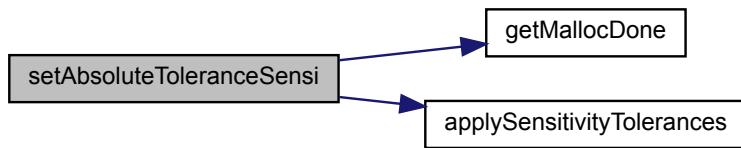
```
void setAbsoluteToleranceSensi (
    double atol )
```

Parameters

atol	absolute tolerance (non-negative number)
------	--

Definition at line 569 of file solver.cpp.

Here is the call graph for this function:



10.34.3.39 getRelativeToleranceQuadratures()

```
double getRelativeToleranceQuadratures ( ) const
```

Returns

relative tolerance

Definition at line 580 of file solver.cpp.

10.34.3.40 setRelativeToleranceQuadratures()

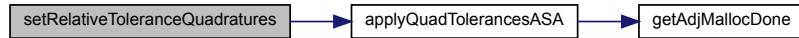
```
void setRelativeToleranceQuadratures (
    double rtol )
```

Parameters

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 584 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.34.3.41 getAbsoluteToleranceQuadratures()**

```
double getAbsoluteToleranceQuadratures ( ) const
```

Returns

absolute tolerance

Definition at line 598 of file solver.cpp.

10.34.3.42 setAbsoluteToleranceQuadratures()

```
void setAbsoluteToleranceQuadratures (
    double atol )
```

Parameters

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 602 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.43 getRelativeToleranceSteadyState()

```
double getRelativeToleranceSteadyState ( ) const
```

Returns

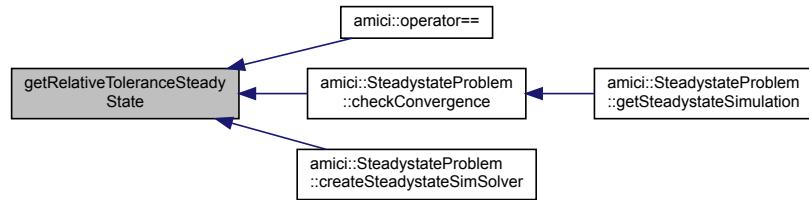
relative tolerance

Definition at line 616 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.44 setRelativeToleranceSteadyState()

```
void setRelativeToleranceSteadyState (
    double rtol )
```

Parameters

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 620 of file solver.cpp.

10.34.3.45 getAbsoluteToleranceSteadyState()

```
double getAbsoluteToleranceSteadyState ( ) const
```

Returns

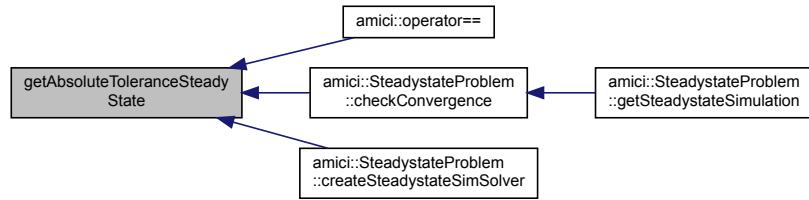
absolute tolerance

Definition at line 627 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.46 setAbsoluteToleranceSteadyState()

```
void setAbsoluteToleranceSteadyState (
    double atol )
```

Parameters

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 631 of file solver.cpp.

10.34.3.47 getRelativeToleranceSteadyStateSensi()

```
double getRelativeToleranceSteadyStateSensi ( ) const
```

Returns

relative tolerance

Definition at line 638 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.48 setRelativeToleranceSteadyStateSensi()

```
void setRelativeToleranceSteadyStateSensi ( double rtol )
```

Parameters

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 642 of file solver.cpp.

10.34.3.49 getAbsoluteToleranceSteadyStateSensi()

```
double getAbsoluteToleranceSteadyStateSensi ( ) const
```

Returns

absolute tolerance

Definition at line 649 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.50 setAbsoluteToleranceSteadyStateSensi()

```
void setAbsoluteToleranceSteadyStateSensi (
    double atol )
```

Parameters

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 653 of file solver.cpp.

10.34.3.51 getMaxSteps()

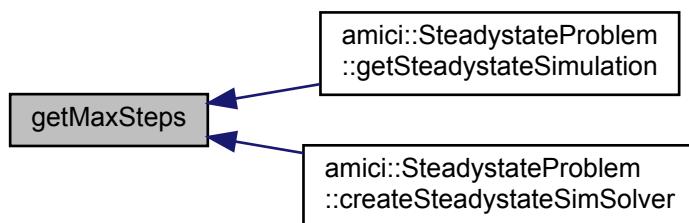
```
int getMaxSteps ( ) const
```

Returns

maximum number of solver steps

Definition at line 660 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.52 setMaxSteps()

```
void setMaxSteps (
    int maxsteps )
```

Parameters

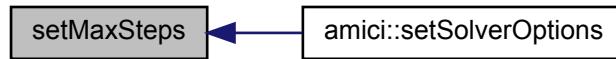
<i>maxsteps</i>	maximum number of solver steps (non-negative number)
-----------------	--

Definition at line 664 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.34.3.53 getMaxStepsBackwardProblem()**

```
int getMaxStepsBackwardProblem ( ) const
```

Returns

maximum number of solver steps

Definition at line 673 of file solver.cpp.

10.34.3.54 setMaxStepsBackwardProblem()

```
void setMaxStepsBackwardProblem (
    int maxsteps )
```

Parameters

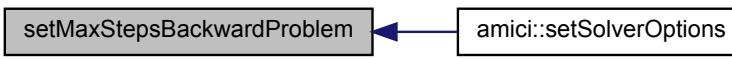
<code>maxsteps</code>	maximum number of solver steps (non-negative number)
-----------------------	--

Definition at line 677 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.34.3.55 getLinearMultistepMethod()**

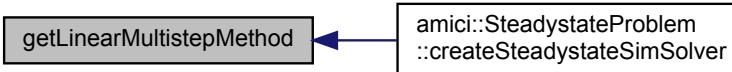
`LinearMultistepMethod getLinearMultistepMethod () const`

Returns

linear system multistep method

Definition at line 687 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.56 setLinearMultistepMethod()**

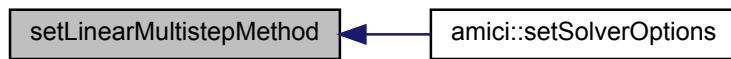
```
void setLinearMultistepMethod (
    LinearMultistepMethod lmm )
```

Parameters

<i>lmm</i>	linear system multistep method
------------	--------------------------------

Definition at line 691 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.57 getNonlinearSolverIteration()**

```
NonlinearSolverIteration getNonlinearSolverIteration ( ) const
```

Returns

Definition at line 697 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.58 setNonlinearSolverIteration()**

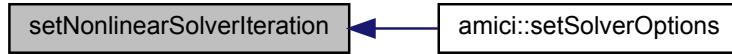
```
void setNonlinearSolverIteration (
    NonlinearSolverIteration iter )
```

Parameters

<i>iter</i>	nonlinear system solution method
-------------	----------------------------------

Definition at line 701 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.59 getInterpolationType()

```
InterpolationType getInterpolationType ( ) const
```

Returns

Definition at line 707 of file solver.cpp.

10.34.3.60 setInterpolationType()

```
void setInterpolationType (
    InterpolationType interpType )
```

Parameters

<i>interpType</i>	interpolation type
-------------------	--------------------

Definition at line 711 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.61 getStateOrdering()

```
StateOrdering getStateOrdering ( ) const
```

Returns

Definition at line 717 of file solver.cpp.

10.34.3.62 setStateOrdering()

```
void setStateOrdering (
    StateOrdering ordering )
```

Parameters

<i>ordering</i>	state ordering
-----------------	----------------

Definition at line 721 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.63 getStabilityLimitFlag()**

```
int getStabilityLimitFlag ( ) const
```

Returns

std::det can be amici.FALSE (deactivated) or amici.TRUE (activated)

Definition at line 731 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.64 setStabilityLimitFlag()

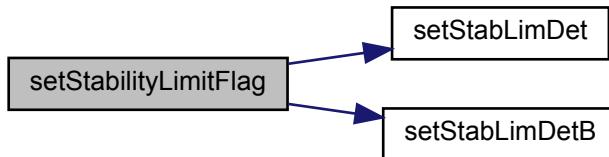
```
void setStabilityLimitFlag (
    booleantype stldet )
```

Parameters

stldet	can be amici.FALSE (deactivated) or amici.TRUE (activated)
--------	--

Definition at line 735 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



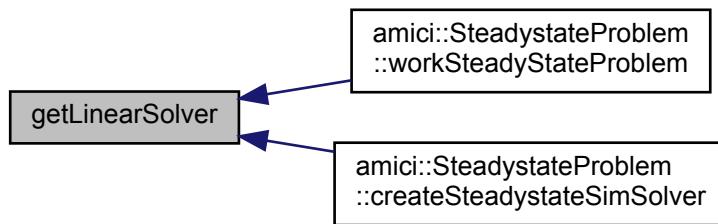
10.34.3.65 getLinearSolver()

```
LinearSolver getLinearSolver ( ) const
```

Returns

Definition at line 748 of file solver.cpp.

Here is the caller graph for this function:

**10.34.3.66 setLinearSolver()**

```
void setLinearSolver (  
    LinearSolver linsol )
```

Parameters

<i>linsol</i>	<input type="text"/>
---------------	----------------------

Definition at line 752 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.67 getInternalSensitivityMethod()

```
InternalSensitivityMethod getInternalSensitivityMethod () const
```

Returns

internal sensitivity method

Definition at line 760 of file solver.cpp.

10.34.3.68 setInternalSensitivityMethod()

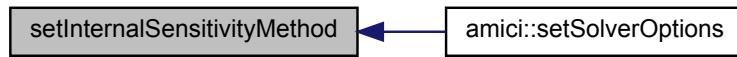
```
void setInternalSensitivityMethod (
    InternalSensitivityMethod ism )
```

Parameters

<i>ism</i>	internal sensitivity method
------------	-----------------------------

Definition at line 764 of file solver.cpp.

Here is the caller graph for this function:



10.34.3.69 init()

```
virtual void init (
    AmiVector * x,
    AmiVector * dx,
    realtype t ) [protected], [pure virtual]
```

init initialises the states at the specified initial timepoint

Parameters

<i>x</i>	initial state variables
<i>dx</i>	initial derivative state variables (DAE only)
<i>t</i>	initial timepoint

Here is the caller graph for this function:



10.34.3.70 binit()

```

virtual void binit (
    int which,
    AmiVector * xB,
    AmiVector * dxB,
    realtype t ) [protected], [pure virtual]
  
```

binit initialises the adjoint states at the specified final timepoint

Parameters

<i>which</i>	identifier of the backwards problem
<i>xB</i>	initial adjoint state variables
<i>dxB</i>	initial adjoint derivative state variables (DAE only)
<i>t</i>	final timepoint

Here is the caller graph for this function:



10.34.3.71 qbinit()

```

virtual void qbinit (
    int which,
    AmiVector * qBdot ) [protected], [pure virtual]
  
```

qbinit initialises the quadrature states at the specified final timepoint

Parameters

<i>which</i>	identifier of the backwards problem
<i>qBdot</i>	initial adjoint quadrature state variables

Here is the caller graph for this function:

**10.34.3.72 rootInit()**

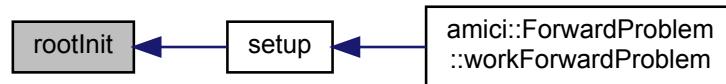
```
virtual void rootInit (
    int ne ) [protected], [pure virtual]
```

RootInit initialises the rootfinding for events

Parameters

<i>ne</i>	number of different events
-----------	----------------------------

Here is the caller graph for this function:

**10.34.3.73 sensInit1()**

```
virtual void sensInit1 (
    AmiVectorArray * sx,
    AmiVectorArray * sdx,
    int nplist ) [protected], [pure virtual]
```

SensInit1 initialises the sensitivities at the specified initial timepoint

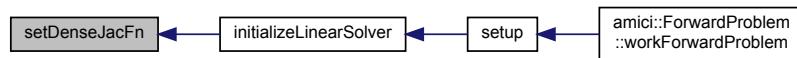
Parameters

<i>sx</i>	initial state sensitivities
<i>sdx</i>	initial derivative state sensitivities (DAE only)
<i>nplist</i>	number parameter wrt which sensitivities are to be computed

10.34.3.74 setDenseJacFn()

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

SetDenseJacFn sets the dense Jacobian function Here is the caller graph for this function:

**10.34.3.75 setSparseJacFn()**

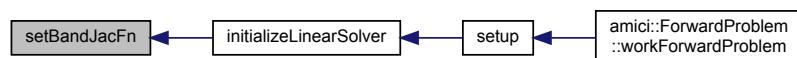
```
virtual void setSparseJacFn ( ) [protected], [pure virtual]
```

SetSparseJacFn sets the sparse Jacobian function

10.34.3.76 setBandJacFn()

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

SetBandJacFn sets the banded Jacobian function Here is the caller graph for this function:

**10.34.3.77 setJacTimesVecFn()**

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

SetJacTimesVecFn sets the Jacobian vector multiplication function

10.34.3.78 setDenseJacFnB()

```
virtual void setDenseJacFnB (
    int which ) [protected], [pure virtual]
```

SetDenseJacFn sets the dense Jacobian function

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Here is the caller graph for this function:

**10.34.3.79 setSparseJacFnB()**

```
virtual void setSparseJacFnB (
    int which ) [protected], [pure virtual]
```

SetSparseJacFn sets the sparse Jacobian function

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

10.34.3.80 setBandJacFnB()

```
virtual void setBandJacFnB (
    int which ) [protected], [pure virtual]
```

SetBandJacFn sets the banded Jacobian function

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Here is the caller graph for this function:



10.34.3.81 setJacTimesVecFnB()

```
virtual void setJacTimesVecFnB (
    int which ) [protected], [pure virtual]
```

SetJacTimesVecFn sets the Jacobian vector multiplication function

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

10.34.3.82 wrapErrorHandlerFn()

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

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

Parameters

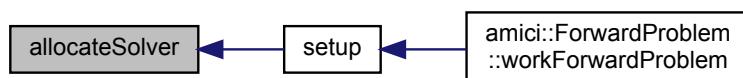
<i>error_code</i>	error identifier
<i>module</i>	name of the module in which the error occurred
<i>function</i>	name of the function in which the error occurred Type: char
<i>msg</i>	error message
<i>eh_data</i>	unused input

Definition at line 149 of file solver.cpp.

10.34.3.83 allocateSolver()

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

Create specifies solver method and initializes solver memory for the forward problem Here is the caller graph for this function:



10.34.3.84 setSStolerances()

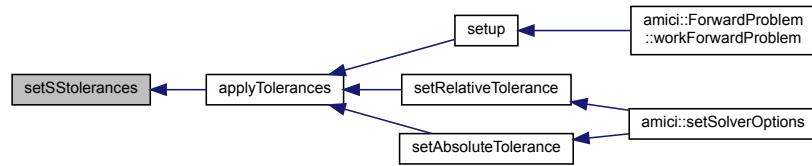
```
virtual void setSStolerances (
    double rtol,
    double atol ) [protected], [pure virtual]
```

SStolerances sets scalar relative and absolute tolerances for the forward problem

Parameters

<i>rtol</i>	relative tolerances
<i>atol</i>	absolute tolerances

Here is the caller graph for this function:



10.34.3.85 setSensSStolerances()

```
virtual void setSensSStolerances (
    double rtol,
    double * atol ) [protected], [pure virtual]
```

SensSStolerances activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

<i>rtol</i>	relative tolerances
<i>atol</i>	array of absolute tolerances for every sensitiv variable

10.34.3.86 setSensErrCon()

```
virtual void setSensErrCon (
    bool error_corr ) [protected], [pure virtual]
```

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

Parameters

<i>error_corr</i>	activation flag
-------------------	-----------------

10.34.3.87 setQuadErrConB()

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

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

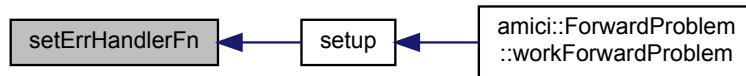
Parameters

<i>which</i>	identifier of the backwards problem
<i>flag</i>	activation flag

10.34.3.88 setErrorHandlerFn()

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

SetErrorHandlerFn attaches the error handler function (errMsgIdAndTxt) to the solver. Here is the caller graph for this function:



10.34.3.89 setUserData()

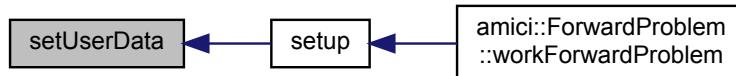
```
virtual void setUserData (
    Model * model ) [protected], [pure virtual]
```

SetUserData attaches the user data instance (here this is a [Model](#)) to the forward problem

Parameters

<i>model</i>	Model instance,
--------------	---------------------------------

Here is the caller graph for this function:



10.34.3.90 setUserDataB()

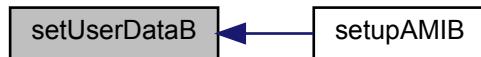
```
virtual void setUserDataB (
    int which,
    Model * model ) [protected], [pure virtual]
```

SetUserDataB attaches the user data instance (here this is a [Model](#)) to the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>model</i>	Model instance,

Here is the caller graph for this function:



10.34.3.91 setMaxNumSteps()

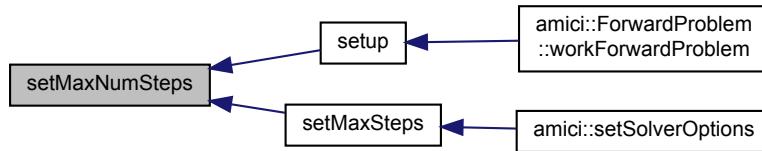
```
virtual void setMaxNumSteps (
    long int mxsteps ) [protected], [pure virtual]
```

SetMaxNumSteps specifies the maximum number of steps for the forward problem

Parameters

<i>mxsteps</i>	number of steps
----------------	-----------------

Here is the caller graph for this function:



10.34.3.92 setMaxNumStepsB()

```

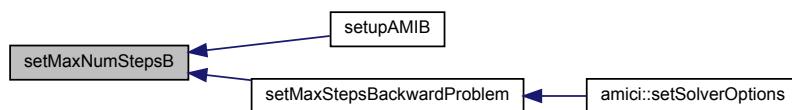
virtual void setMaxNumStepsB (
    int which,
    long int mxstepsB ) [protected], [pure virtual]
  
```

SetMaxNumStepsB specifies the maximum number of steps for the forward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>mxstepsB</i>	number of steps

Here is the caller graph for this function:



10.34.3.93 setStabLimDet()

```

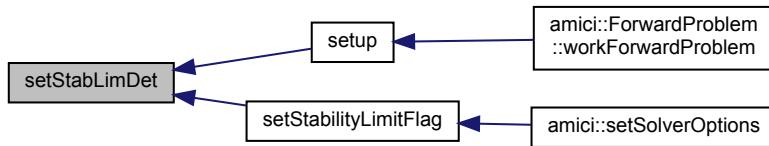
virtual void setStabLimDet (
    int stldet ) [protected], [pure virtual]
  
```

SetStabLimDet activates stability limit detection for the forward problem

Parameters

<i>stldet</i>	flag for stability limit detection (TRUE or FALSE)
---------------	--

Here is the caller graph for this function:



10.34.3.94 setStabLimDetB()

```

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

SetStabLimDetB activates stability limit detection for the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>stldet</i>	flag for stability limit detection (TRUE or FALSE)

Here is the caller graph for this function:



10.34.3.95 setId()

```

virtual void setId (
    Model * model ) [protected], [pure virtual]
  
```

SetId specify algebraic/differential components (DAE only)

Parameters

<i>model</i>	model specification
--------------	---------------------

10.34.3.96 setSuppressAlg()

```
virtual void setSuppressAlg (
    bool flag ) [protected], [pure virtual]
```

SetId deactivates error control for algebraic components (DAE only)

Parameters

<i>flag</i>	deactivation flag
-------------	-------------------

10.34.3.97 setSensParams()

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

SetSensParams specifies the scaling and indexes for sensitivity computation

Parameters

<i>p</i>	paramters
<i>pbar</i>	parameter scaling constants
<i>plist</i>	parameter index list

10.34.3.98 getDky()

```
virtual void getDky (
    realtype t,
    int k,
    AmiVector * dky ) const [protected], [pure virtual]
```

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

Parameters

<i>t</i>	timepoint
<i>k</i>	derivative order
<i>dky</i>	interpolated solution

10.34.3.99 adjInit()

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

AdjInit initializes the adjoint problem

10.34.3.100 allocateSolverB()

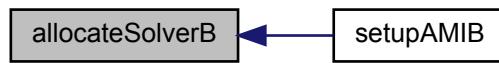
```
virtual void allocateSolverB (
    int * which ) [protected], [pure virtual]
```

specifies solver method and initializes solver memory for the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Here is the caller graph for this function:



10.34.3.101 setSStolerancesB()

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

SStolerancesB sets relative and absolute tolerances for the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>relTolB</i>	relative tolerances
<i>absTolB</i>	absolute tolerances

10.34.3.102 quadSStolerancesB()

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

SStolerancesB sets relative and absolute tolerances for the quadrature backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>reltolQB</i>	relative tolerances
<i>abstolQB</i>	absolute tolerances

10.34.3.103 dense()

```
virtual void dense (
    int nx ) [protected], [pure virtual]
```

Dense attaches a dense linear solver to the forward problem

Parameters

<i>nx</i>	number of state variables
-----------	---------------------------

Here is the caller graph for this function:

**10.34.3.104 denseB()**

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

DenseB attaches a dense linear solver to the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>nx</i>	number of state variables

Here is the caller graph for this function:



10.34.3.105 band()

```

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

Band attaches a banded linear solver to the forward problem

Parameters

<i>nx</i>	number of state variables
<i>ubw</i>	upper matrix bandwidth
<i>lbw</i>	lower matrix bandwidth

Here is the caller graph for this function:



10.34.3.106 bandB()

```

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

BandB attaches a banded linear solver to the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>nx</i>	number of state variables
<i>ubw</i>	upper matrix bandwidth
<i>lbw</i>	lower matrix bandwidth

Here is the caller graph for this function:



10.34.3.107 diag()

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

Diag attaches a diagonal linear solver to the forward problem

10.34.3.108 diagB()

```
virtual void diagB (
    int which ) [protected], [pure virtual]
```

DiagB attaches a diagonal linear solver to the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

10.34.3.109 spgmr()

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

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

Parameters

<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.110 spgmrB()

```
virtual void spgmrB (
    int which,
```

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

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

Parameters

<i>which</i>	identifier of the backwards problem
<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.111 spbcg()

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

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

Parameters

<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.112 spbcgB()

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

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

Parameters

<i>which</i>	identifier of the backwards problem
<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.113 sptfqmr()

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

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

Parameters

<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.114 sptfqmrB()

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

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

Parameters

<i>which</i>	identifier of the backwards problem
<i>prectype</i>	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
<i>maxl</i>	maximum Kryloc subspace dimension

10.34.3.115 klu()

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

KLU attaches a sparse linear solver to the forward problem

Parameters

<i>nx</i>	number of state variables
<i>nnz</i>	number of nonzero entries in the jacobian
<i>sparsetype</i>	sparse storage type, CSC_MAT for column matrix, CSR_MAT for row matrix

10.34.3.116 kluSetOrdering()

```
virtual void kluSetOrdering (
    int ordering ) [protected], [pure virtual]
```

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

Parameters

<i>ordering</i>	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering
-----------------	--

10.34.3.117 kluSetOrderingB()

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

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

Parameters

<i>which</i>	identifier of the backwards problem
<i>ordering</i>	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering

10.34.3.118 kluB()

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

KLUB attaches a sparse linear solver to the forward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>nx</i>	number of state variables
<i>nnz</i>	number of nonzero entries in the jacobian
<i>sparsetype</i>	sparse storage type, CSC_MAT for column matrix, CSR_MAT for row matrix

10.34.3.119 getNumSteps()

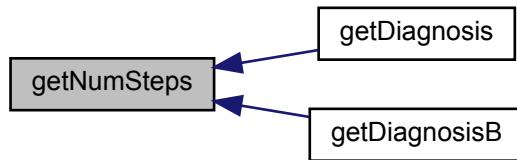
```
virtual void getNumSteps (
    void * ami_mem,
    long int * numsteps) const [protected], [pure virtual]
```

getNumSteps reports the number of solver steps

Parameters

<i>ami_mem</i>	pointer to the solver memory instance (can be from forward or backward problem)
<i>numsteps</i>	output array

Here is the caller graph for this function:



10.34.3.120 getNumRhsEvals()

```

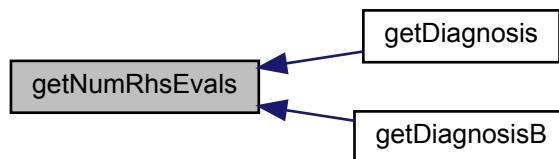
virtual void getNumRhsEvals (
    void * ami_mem,
    long int * numrhsvals ) const [protected], [pure virtual]
  
```

getNumRhsEvals reports the number of right hand evaluations

Parameters

<i>ami_mem</i>	pointer to the solver memory instance (can be from forward or backward problem)
<i>numrhsvals</i>	output array

Here is the caller graph for this function:



10.34.3.121 getNumErrTestFails()

```

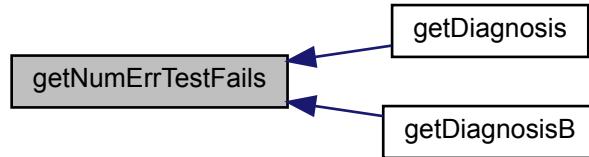
virtual void getNumErrTestFails (
    void * ami_mem,
    long int * numerertestfails ) const [protected], [pure virtual]
  
```

getNumErrTestFails reports the number of local error test failures

Parameters

<i>ami_mem</i>	pointer to the solver memory instance (can be from forward or backward problem)
<i>numerrtestfails</i>	output array

Here is the caller graph for this function:

**10.34.3.122 getNumNonlinSolvConvFails()**

```

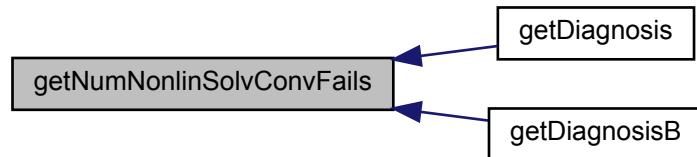
virtual void getNumNonlinSolvConvFails (
    void * ami_mem,
    long int * numnonlinsolvconvfails ) const [protected], [pure virtual]
  
```

getNumNonlinSolvConvFails reports the number of nonlinear convergence failures

Parameters

<i>ami_mem</i>	pointer to the solver memory instance (can be from forward or backward problem)
<i>numnonlinsolvconvfails</i>	output array

Here is the caller graph for this function:



10.34.3.123 getLastOrder()

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

Reports the order of the integration method during the last internal step

Parameters

<i>ami_mem</i>	pointer to the solver memory instance (can be from forward or backward problem)
<i>order</i>	output array

Here is the caller graph for this function:

**10.34.3.124 initializeLinearSolver()**

```
void initializeLinearSolver (
    const Model * model ) [protected]
```

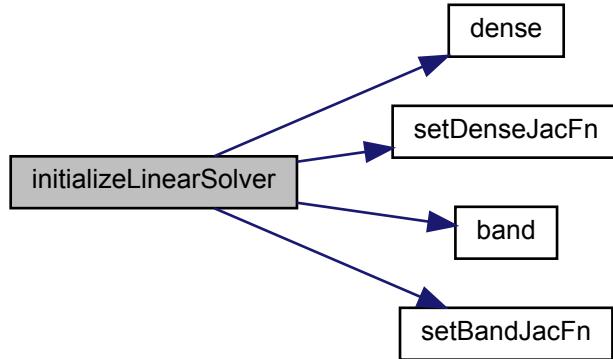
initializeLinearSolver sets the linear solver for the forward problem

Parameters

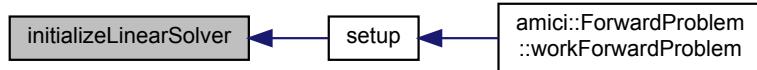
<i>model</i>	pointer to the model object
--------------	-----------------------------

Definition at line 227 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.125 initializeLinearSolverB()

```

void initializeLinearSolverB (
    const Model * model,
    const int which ) [protected]
  
```

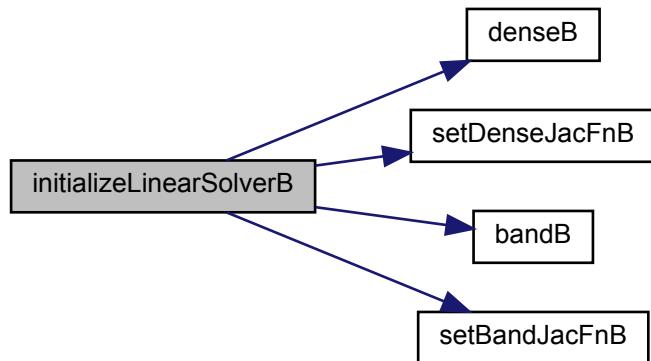
Sets the linear solver for the backward problem

Parameters

<code>model</code>	pointer to the model object
<code>which</code>	index of the backward problem

Definition at line 304 of file `solver.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.126 nplist()

```
virtual int nplist ( ) const [protected], [pure virtual]
```

Accessor function to the number of sensitivity parameters in the model stored in the user data

Returns

number of sensitivity parameters

10.34.3.127 nx()

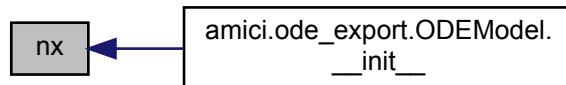
```
virtual int nx ( ) const [protected], [pure virtual]
```

Accessor function to the number of state variables in the model stored in the user data

Returns

number of state variables

Here is the caller graph for this function:

**10.34.3.128 getModel()**

```
virtual const Model* getModel ( ) const [protected], [pure virtual]
```

Accessor function to the model stored in the user data

Returns

user data model

10.34.3.129 getMallocDone()

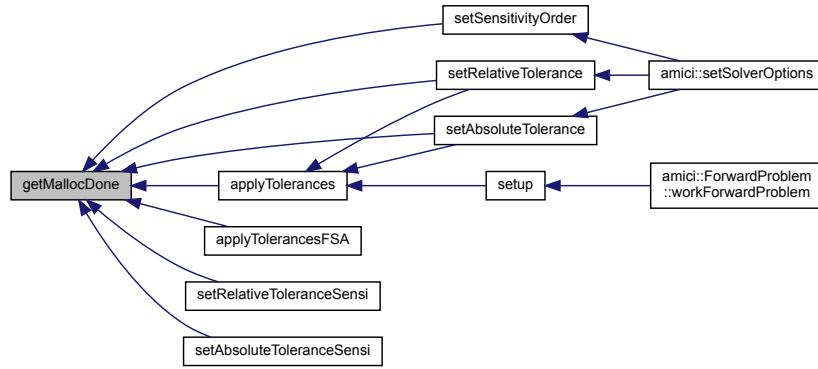
```
virtual bool getMallocDone ( ) const [protected], [pure virtual]
```

checks whether memory for the forward problem has been allocated

Returns

```
solverMemory->(cv|ida)___MallocDone
```

Here is the caller graph for this function:

**10.34.3.130 getAdjMallocDone()**

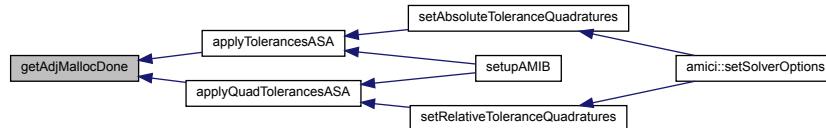
```
virtual bool getAdjMallocDone ( ) const [protected], [pure virtual]
```

checks whether memory for the backward problem has been allocated

Returns

```
solverMemory->(cv|ida)___adjMallocDone
```

Here is the caller graph for this function:

**10.34.3.131 getAdjBmem()**

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

`getAdjBmem` retrieves the solver memory instance for the backward problem

Parameters

<i>which</i>	identifier of the backwards problem
<i>ami_mem</i>	pointer to the forward solver memory instance

Returns

ami_memB pointer to the backward solver memory instance

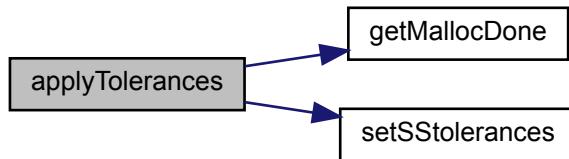
10.34.3.132 applyTolerances()

```
void applyTolerances ( ) [protected]
```

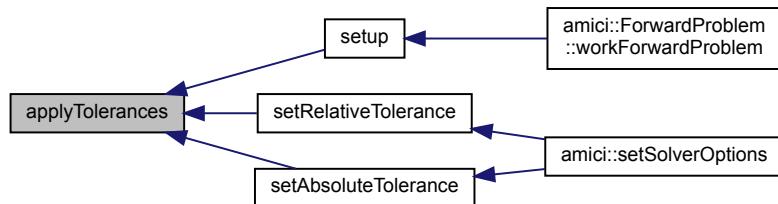
updates solver tolerances according to the currently specified member variables

Definition at line 408 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



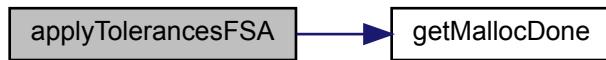
10.34.3.133 applyTolerancesFSA()

```
void applyTolerancesFSA ( ) [protected]
```

updates FSA solver tolerances according to the currently specified member variables

Definition at line 415 of file solver.cpp.

Here is the call graph for this function:

**10.34.3.134 applyTolerancesASA()**

```
void applyTolerancesASA ( int which ) [protected]
```

updates ASA solver tolerances according to the currently specified member variables

Parameters

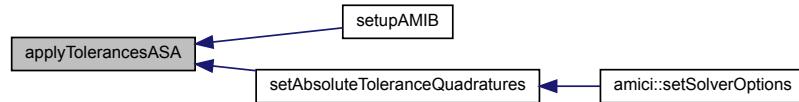
<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Definition at line 429 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.34.3.135 applyQuadTolerancesASA()

```
void applyQuadTolerancesASA (
    int which ) [protected]
```

updates ASA quadrature solver tolerances according to the currently specified member variables

Parameters

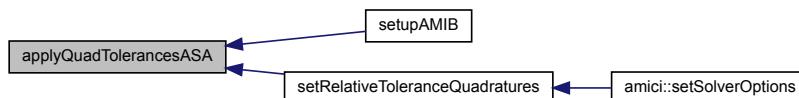
<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Definition at line 440 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



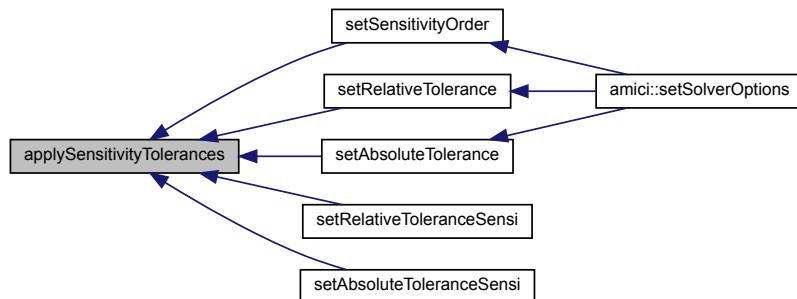
10.34.3.136 applySensitivityTolerances()

```
void applySensitivityTolerances ( ) [protected]
```

updates all sensitivity solver tolerances according to the currently specified member variables

Definition at line 459 of file solver.cpp.

Here is the caller graph for this function:



10.34.4 Friends And Related Function Documentation

10.34.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
    Archive & ar,
    Solver & r,
    const unsigned int version ) [friend]
```

Parameters

<code>ar</code>	Archive to serialize to
<code>r</code>	Data to serialize
<code>version</code>	Version number

10.34.4.2 operator==

```
bool operator== (
    const Solver & a,
    const Solver & b ) [friend]
```

Parameters

<code>a</code>	
<code>b</code>	

Returns

Definition at line 378 of file solver.cpp.

10.34.5 Member Data Documentation**10.34.5.1 solverMemory**

```
std::unique_ptr<void, std::function<void(void *)> > solverMemory [protected]
```

pointer to solver memory block

Definition at line 1107 of file solver.h.

10.34.5.2 solverMemoryB

```
std::vector<std::unique_ptr<void, std::function<void(void *)> > > solverMemoryB [protected]
```

pointer to solver memory block

Definition at line 1110 of file solver.h.

10.34.5.3 solverWasCalled

```
bool solverWasCalled = false [protected]
```

flag indicating whether the solver was called

Definition at line 1113 of file solver.h.

10.34.5.4 ism

```
InternalSensitivityMethod ism = InternalSensitivityMethod::simultaneous [protected]
```

internal sensitivity method flag used to select the sensitivity solution method. Only applies for Forward Sensitivities.

Definition at line 1117 of file solver.h.

10.34.5.5 lmm

```
LinearMultistepMethod lmm = LinearMultistepMethod::BDF [protected]
```

specifies the linear multistep method.

Definition at line 1121 of file solver.h.

10.34.5.6 iter

```
NonlinearSolverIteration iter = NonlinearSolverIteration::newton [protected]
```

specifies the type of nonlinear solver iteration

Definition at line 1126 of file solver.h.

10.34.5.7 interpType

```
InterpolationType interpType = InterpolationType::hermite [protected]
```

interpolation type for the forward problem solution which is then used for the backwards problem.

Definition at line 1131 of file solver.h.

10.34.5.8 maxsteps

```
int maxsteps = 10000 [protected]
```

maximum number of allowed integration steps

Definition at line 1134 of file solver.h.

10.35 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>
```

Public Member Functions

- void [workSteadyStateProblem](#) ([ReturnData](#) *rdata, [Solver](#) *solver, [Model](#) *model, int it)
- [realtype](#) [getWrmsNorm](#) ([AmiVector](#) const &x, [AmiVector](#) const &xdot, [realtype](#) atol, [realtype](#) rtol)
- bool [checkConvergence](#) (const [Solver](#) *solver, [Model](#) *model)
- void [applyNewtonsMethod](#) ([ReturnData](#) *rdata, [Model](#) *model, [NewtonSolver](#) *newtonSolver, int newton_try)
- void [writeNewtonOutput](#) ([ReturnData](#) *rdata, const [Model](#) *model, [NewtonStatus](#) newton_status, double run_time, int it)
- void [getSteadystateSimulation](#) ([ReturnData](#) *rdata, [Solver](#) *solver, [Model](#) *model, int it)
- std::unique_ptr<[CVodeSolver](#)> [createSteadystateSimSolver](#) ([Solver](#) *solver, [Model](#) *model, [realtype](#) tstart)
- [SteadystateProblem](#) ([realtype](#) *t, [AmiVector](#) *x, [AmiVectorArray](#) *sx)

10.35.1 Detailed Description

Definition at line 25 of file steadystateproblem.h.

10.35.2 Constructor & Destructor Documentation

10.35.2.1 SteadyStateProblem()

```
SteadyStateProblem (
    realtype * t,
    AmiVector * x,
    AmiVectorArray * sx )
```

default constructor

Parameters

<i>t</i>	pointer to time variable
<i>x</i>	pointer to state variables
<i>sx</i>	pointer to state sensitivity variables

Definition at line 108 of file steadystateproblem.h.

10.35.3 Member Function Documentation

10.35.3.1 workSteadyStateProblem()

```
void workSteadyStateProblem (
    ReturnData * rdata,
    Solver * solver,
    Model * model,
    int it )
```

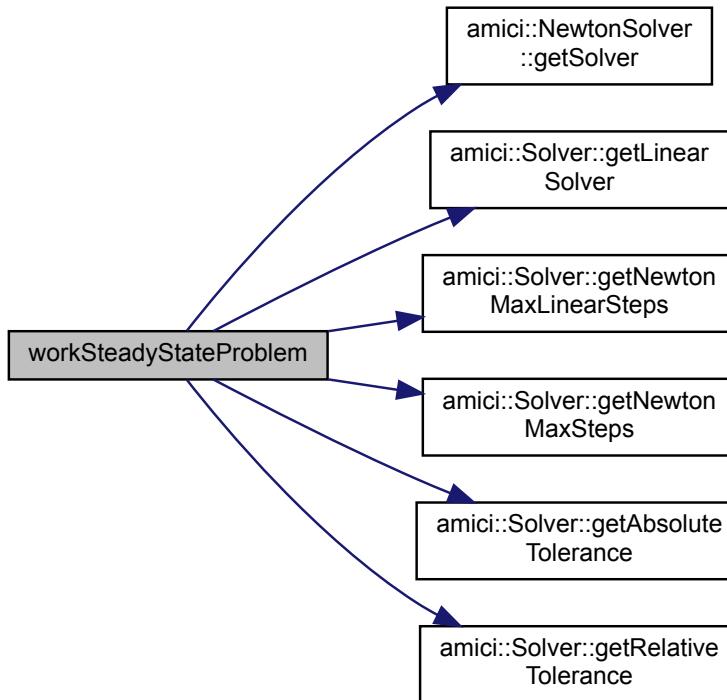
Tries to determine the steady state of the ODE system by a Newton solver, uses forward intergration, if the Newton solver fails, restarts Newton solver, if integration fails. Computes steady state sensitivities

Parameters

<i>solver</i>	pointer to the AMICI solver object
<i>model</i>	pointer to the AMICI model object
<i>it</i>	integer with the index of the current time step
<i>rdata</i>	pointer to the return data object

Definition at line 21 of file steadystateproblem.cpp.

Here is the call graph for this function:



10.35.3.2 getWrmsNorm()

```

realtype getWrmsNorm (
    AmiVector const & x,
    AmiVector const & xdot,
    realtype atol,
    realtype rtol )
  
```

Computes the weighted root mean square of `xdot` the weights are computed according to `x`: $w_i = 1 / (rtol * x_i + atol)$

Parameters

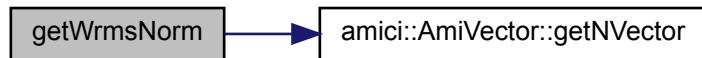
<code>x</code>	current state
<code>xdot</code>	current rhs
<code>atol</code>	absolute tolerance
<code>rtol</code>	relative tolerance

Returns

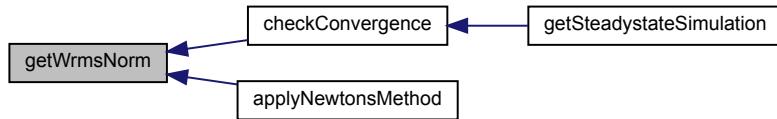
root-mean-square norm

Definition at line 95 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.35.3.3 checkConvergence()

```
bool checkConvergence (
    const Solver * solver,
    Model * model )
```

Checks convergence for state and respective sensitivities

Parameters

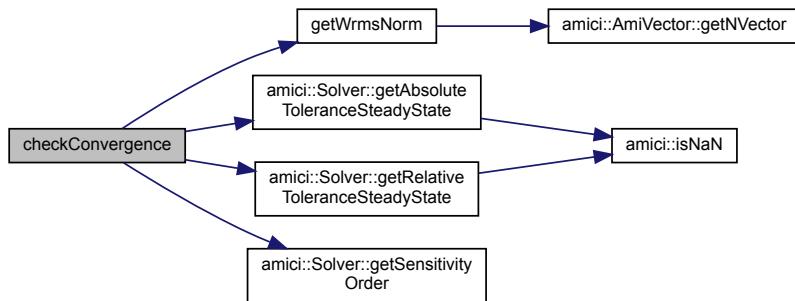
<i>solver</i>	Solver instance
<i>model</i>	instance

Returns

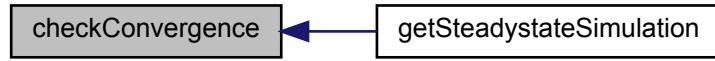
boolean indicating convergence

Definition at line 107 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



10.35.3.4 applyNewtonsMethod()

```

void applyNewtonsMethod (
    ReturnData * rdata,
    Model * model,
    NewtonSolver * newtonSolver,
    int newton_try )
  
```

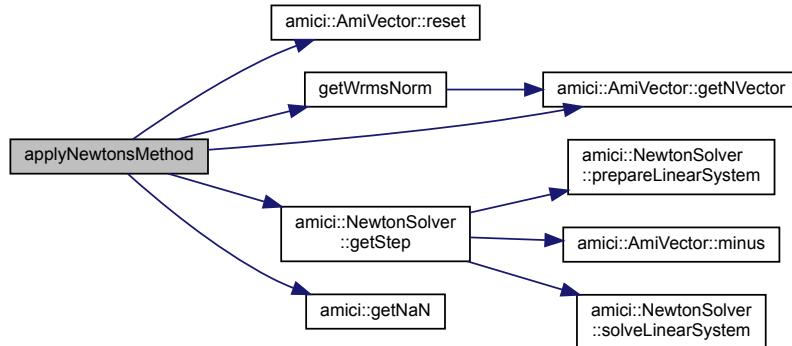
Runs the Newton solver iterations and checks for convergence to steady state

Parameters

<i>rdata</i>	pointer to the return data object
<i>model</i>	pointer to the AMICI model object
<i>newtonSolver</i>	pointer to the NewtonSolver object Type: NewtonSolver
<i>newton_try</i>	integer start number of Newton solver (1 or 2)

Definition at line 132 of file `steadystateproblem.cpp`.

Here is the call graph for this function:



10.35.3.5 writeNewtonOutput()

```
void writeNewtonOutput (
    ReturnData * rdata,
    const Model * model,
    NewtonStatus newton_status,
    double run_time,
    int it )
```

Stores output of workSteadyStateProblem in return data

Parameters

<i>newton_status</i>	integer flag indicating when a steady state was found
<i>run_time</i>	double coputation time of the solver in milliseconds
<i>rdata</i>	pointer to the return data instance
<i>model</i>	pointer to the model instance
<i>it</i>	current timepoint index, <0 indicates preequilibration

Definition at line 221 of file steadystateproblem.cpp.

10.35.3.6 getSteadystateSimulation()

```
void getSteadystateSimulation (
    ReturnData * rdata,
    Solver * solver,
    Model * model,
    int it )
```

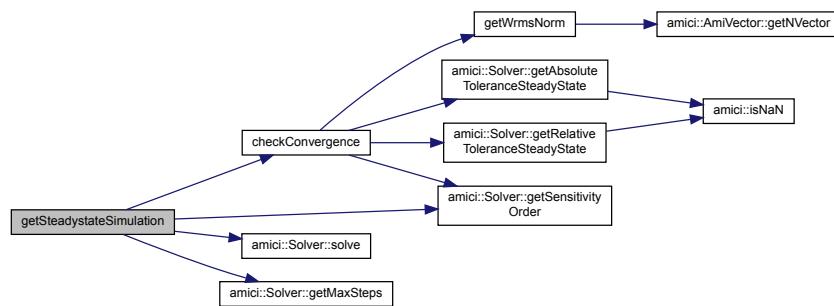
Forward simulation is launched, if Newton solver fails in first try

Parameters

<i>solver</i>	pointer to the AMICI solver object
<i>model</i>	pointer to the AMICI model object
<i>rdata</i>	pointer to the return data object
<i>it</i>	current timepoint index, <0 indicates preequilibration

Definition at line 246 of file steadystateproblem.cpp.

Here is the call graph for this function:

**10.35.3.7 createSteadystateSimSolver()**

```
std::unique_ptr< CVodeSolver > createSteadystateSimSolver (
    Solver * solver,
    Model * model,
    realscalar tstart )
```

initialize CVodeSolver instance for preequilibration simulation

Parameters

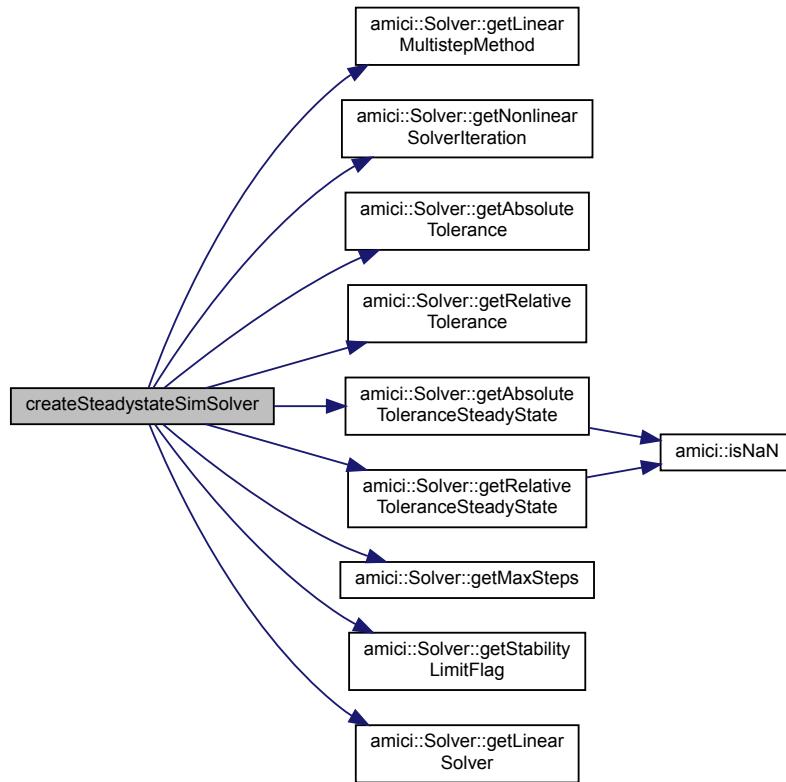
<i>solver</i>	pointer to the AMICI solver object
<i>model</i>	pointer to the AMICI model object
<i>tstart</i>	time point for starting Newton simulation

Returns

solver instance

Definition at line 273 of file steadystateproblem.cpp.

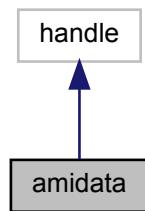
Here is the call graph for this function:



10.36 amidata Class Reference

AMIDATA provides a data container to pass experimental data to the simulation routine for likelihood computation. When any of the properties are updated, the class automatically checks consistency of dimension and updates related properties and initialises them with NaNs.

Inheritance diagram for amidata:



Public Member Functions

- `amidata` (matlabtypesubstitute varargin)

amidata creates an amidata container for experimental data with specified dimensions amidata.

Public Attributes

- matlabtypesubstitute `nt` = 0
number of timepoints
- matlabtypesubstitute `ny` = 0
number of observables
- matlabtypesubstitute `nz` = 0
number of event observables
- matlabtypesubstitute `ne` = 0
number of events
- matlabtypesubstitute `nk` = 0
number of conditions/constants
- matlabtypesubstitute `t` = double.empty("")
timepoints of observations
- matlabtypesubstitute `Y` = double.empty("")
observations
- matlabtypesubstitute `Sigma_Y` = double.empty("")
standard deviation of observations
- matlabtypesubstitute `Z` = double.empty("")
event observations
- matlabtypesubstitute `Sigma_Z` = double.empty("")
standard deviation of event observations
- matlabtypesubstitute `condition` = double.empty("")
experimental condition
- matlabtypesubstitute `conditionPreequilibration` = double.empty("")
experimental condition for preequilibration

10.36.1 Detailed Description

Definition at line 17 of file amidata.m.

10.36.2 Constructor & Destructor Documentation

10.36.2.1 amidata()

```
amidata (
    matlabtypesubstitute varargin )
```

AMIDATA(amidata) creates a copy of the input container

AMIDATA(struct) tries to creates an amidata container from the input struct. the struct should have the following

fields

`t` [nt,1] `Y` [nt,ny] `Sigma_Y` [nt,ny] `Z` [ne,nz] `Sigma_Z` [ne,nz] `condition` [nk,1] `conditionPreequilibration` [nk,1] if some fields are missing the function will try to initialise them with NaNs with consistent dimensions

AMIDATA(nt,ny,nz,ne,nk) constructs an empty data container with in the provided dimensions intialised with NaNs

Parameters

<i>varargin</i>	<input type="checkbox"/>
-----------------	--------------------------

Definition at line 130 of file amidata.m.

10.36.3 Member Data Documentation**10.36.3.1 nt**

```
nt = 0
```

Default: 0**Note**

This property has custom functionality when its value is changed.

Definition at line 31 of file amidata.m.

10.36.3.2 ny

```
ny = 0
```

Default: 0**Note**

This property has custom functionality when its value is changed.

Definition at line 39 of file amidata.m.

10.36.3.3 nz

```
nz = 0
```

Default: 0**Note**

This property has custom functionality when its value is changed.

Definition at line 47 of file amidata.m.

10.36.3.4 ne

```
ne = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 55 of file amidata.m.

10.36.3.5 nk

```
nk = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 63 of file amidata.m.

10.36.3.6 t

```
t = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 71 of file amidata.m.

10.36.3.7 Y

```
Y = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 79 of file amidata.m.

10.36.3.8 Sigma_Y

```
Sigma_Y = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 87 of file amidata.m.

10.36.3.9 Z

```
Z = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 95 of file amidata.m.

10.36.3.10 Sigma_Z

```
Sigma_Z = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 103 of file amidata.m.

10.36.3.11 condition

```
condition = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 111 of file amidata.m.

10.36.3.12 conditionPreequilibration

```
conditionPreequilibration = double.empty("")
```

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 119 of file amidata.m.

10.37 amievent Class Reference

AMIEVENT defines events which later on will be transformed into appropriate C code.

Public Member Functions

- **amievent** (matlabtypesubstitute **trigger**, matlabtypesubstitute **bolus**, matlabtypesubstitute **z**)
amievent constructs an amievent object from the provided input.
- **mlhsInnerSubst< matlabtypesubstitute > setHflag** (:double **hflag**)
gethflag sets the hflag property.

Public Attributes

- ::symbolic **trigger** = sym.empty("")
the trigger function activates the event on every zero crossing
- ::symbolic **bolus** = sym.empty("")
the bolus function defines the change in states that is applied on every event occurrence
- ::symbolic **z** = sym.empty("")
output function for the event
- matlabtypesubstitute **hflag** = logical.empty("")
flag indicating that a heaviside function is present, this helps to speed up symbolic computations

10.37.1 Detailed Description

Definition at line 17 of file amievent.m.

10.37.2 Constructor & Destructor Documentation

10.37.2.1 amievent()

```
amievent (
    matlabtypesubstitute trigger,
    matlabtypesubstitute bolus,
    matlabtypesubstitute z )
```

Parameters

<i>trigger</i>	trigger function, the event will be triggered on at all roots of this function
<i>bolus</i>	the bolus that will be added to all states on every occurrence of the event
<i>z</i>	the event output that will be reported on every occurrence of the event

Definition at line 75 of file amievent.m.

10.37.3 Member Function Documentation

10.37.3.1 setHflag()

```
mlhsInnerSubst<::amievent> setHflag (
    ::double hflag )
```

Parameters

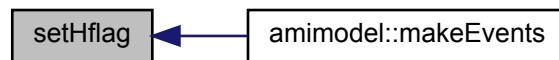
<i>hflag</i>	value for the hflag property
--------------	------------------------------

Return values

<i>this</i>	updated event definition object
-------------	---------------------------------

Definition at line 18 of file setHflag.m.

Here is the caller graph for this function:



10.37.4 Member Data Documentation

10.37.4.1 trigger

```
trigger = sym.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: sym.empty("")

Definition at line 27 of file amievent.m.

10.37.4.2 bolus

```
bolus = sym.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: sym.empty("")

Definition at line 38 of file amievent.m.

10.37.4.3 z

```
z = sym.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: sym.empty("")

Definition at line 49 of file amievent.m.

10.37.4.4 hflag

```
hflag = logical.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: logical.empty("")

Definition at line 60 of file amievent.m.

10.38 amifun Class Reference

AMIFUN defines functions which later on will be transformed into appropriate C code.

Public Member Functions

- **amifun** (matlabtypesubstitute *funstr*, matlabtypesubstitute *model*)

amievent constructs an amifun object from the provided input.
- **noret::substitute writeCcode_sensi** (:**amimodel** *model*,::fileid *fid*)

writeCcode_sensi is a wrapper for writeCcode which loops over parameters and reduces overhead by check nonzero values
- **noret::substitute writeCcode** (:**amimodel** *model*,::fileid *fid*)

writeCcode is a wrapper for gccode which initialises data and reduces overhead by check nonzero values
- **noret::substitute writeMcode** (:**amimodel** *model*)

writeMcode generates matlab evaluable code for specific model functions
- **noret::substitute gccode** (:**amimodel** *model*,::fileid *fid*)

gccode transforms symbolic expressions into c code and writes the respective expression into a specified file
- **mlhsInnerSubst< matlabtypesubstitute > getDeps** (:**amimodel** *model*)

getDeps populates the sensiflag for the requested function
- **mlhsInnerSubst< matlabtypesubstitute > getArgs** (:**amimodel** *model*)

getArgs populates the fargstr property with the argument string of the respective model function (if applicable). model functions are not wrapped versions of functions which have a model specific name and for which the call is solver specific.
- **mlhsInnerSubst< matlabtypesubstitute > getNVecs** ()

getfunargs populates the nvecs property with the names of the N_Vector elements which are required in the execution of the function (if applicable). the information is directly extracted from the argument string
- **mlhsInnerSubst< matlabtypesubstitute > getCVar** ()

getCVar populates the cvar property
- **mlhsInnerSubst< matlabtypesubstitute > getSensiFlag** ()

getSensiFlag populates the sensiflag property
- **mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >,mlhsInnerSubst<::amimodel > > getSyms** (↔ ::**amimodel** *model*)

getSyms computes the symbolic expression for the requested function

Public Attributes

- ::symbolic **sym** = sym("[]")

symbolic definition struct
- ::symbolic **sym_noopt** = sym("[]")

symbolic definition which was not optimized (no dependencies on w)
- ::symbolic **strsym** = sym("[]")

short symbolic string which can be used for the reuse of precomputed values
- ::symbolic **strsym_old** = sym("[]")

short symbolic string which can be used for the reuse of old values
- ::char **funstr** = char.empty("")

name of the model
- ::char **cvar** = char.empty("")

name of the c variable
- ::char **argstr** = char.empty("")

argument string (solver specific)

- ::cell **deps** = cell.empty("")
dependencies on other functions
- matlabypesubstitute **nvecs** = cell.empty("")
nvec dependencies
- matlabypesubstitute **sensiflag** = logical.empty("")
indicates whether the function is a sensitivity or derivative with respect to parameters

10.38.1 Detailed Description

Definition at line 17 of file amifun.m.

10.38.2 Constructor & Destructor Documentation

10.38.2.1 amifun()

```
amifun (
    matlabypesubstitute funstr,
    matlabypesubstitute model )
```

Parameters

<i>funstr</i>	name of the requested function
<i>model</i>	amimodel object which carries all symbolic definitions to construct the function

Definition at line 111 of file amifun.m.

10.38.3 Member Function Documentation

10.38.3.1 writeCcode_sensi()

```
noret::substitute writeCcode_sensi (
    ::amimodel model,
    ::fileid fid )
```

Parameters

<i>model</i>	model defintion object
<i>fid</i>	file id in which the final expression is written

Return values

<i>fid</i>	void
------------	------

Definition at line 18 of file writeCcode_sensi.m.

10.38.3.2 writeCcode()

```
noret::substitute writeCcode (
    ::amimodel model,
    ::fileid fid )
```

Parameters

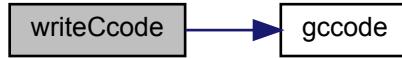
<i>model</i>	model defintion object
<i>fid</i>	file id in which the final expression is written

Return values

<i>fid</i>	void
------------	------

Definition at line 18 of file writeCcode.m.

Here is the call graph for this function:



10.38.3.3 writeMcode()

```
noret::substitute writeMcode (
    ::amimodel model )
```

Parameters

<i>model</i>	model defintion object
--------------	------------------------

Return values

<i>model</i>	void
--------------	------

Definition at line 18 of file writeMcode.m.

10.38.3.4 gccode()

```
mlhsInnerSubst<::amifun> gccode (
    ::amimodel model,
    ::fileid fid )
```

Parameters

<i>model</i>	model definition object
<i>fid</i>	file id in which the expression should be written

Return values

<i>this</i>	function definition object
-------------	----------------------------

Definition at line 18 of file gccode.m.

Here is the caller graph for this function:



10.38.3.5 getDeps()

```
mlhsInnerSubst<::amifun> getDeps (
    ::amimodel model )
```

Parameters

<i>model</i>	model definition object
--------------	-------------------------

Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getDeps.m.

10.38.3.6 getArgs()

```
mlhsInnerSubst<::amifun> getArgs (
    ::amimodel model )
```

Parameters

<i>model</i>	model definition object
--------------	-------------------------

Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getArgs.m.

10.38.3.7 getNVecs()

```
mlhsInnerSubst<::amifun > getNVecs ( )
```

Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getNVecs.m.

10.38.3.8 getCVar()

```
mlhsInnerSubst<::amifun > getCVar ( )
```

Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getCVar.m.

10.38.3.9 getSensiFlag()

```
mlhsInnerSubst<::amifun > getSensiFlag ( )
```

Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getSensiFlag.m.

10.38.3.10 getSyms()

```
mlhsSubst< mlhsInnerSubst<::amifun >, mlhsInnerSubst<::amimodel > > getSyms ( ::amimodel model )
```

Parameters

<i>model</i>	model definition object
--------------	-------------------------

Return values

<i>this</i>	updated function definition object
<i>model</i>	updated model definition object

Definition at line 18 of file getSyms.m.

10.38.4 Member Data Documentation

10.38.4.1 sym

```
sym = sym(" [ ]")
```

Default: sym("[]")

Definition at line 27 of file amifun.m.

10.38.4.2 sym_noopt

```
sym_noopt = sym(" [ ]")
```

Default: sym("[]")

Definition at line 35 of file amifun.m.

10.38.4.3 strsym

```
strsym = sym(" [ ]")
```

Default: sym("[]")

Definition at line 43 of file amifun.m.

10.38.4.4 strsym_old

```
strsym_old = sym("[ ]")
```

Default: `sym("[])")`

Definition at line 51 of file amifun.m.

10.38.4.5 funstr

```
funstr = char.empty("")
```

Default: `char.empty("")`

Definition at line 59 of file amifun.m.

10.38.4.6 cvar

```
cvar = char.empty("")
```

Default: `char.empty("")`

Definition at line 67 of file amifun.m.

10.38.4.7 argstr

```
argstr = char.empty("")
```

Default: `char.empty("")`

Definition at line 75 of file amifun.m.

10.38.4.8 deps

```
deps = cell.empty("")
```

Default: `cell.empty("")`

Definition at line 83 of file amifun.m.

10.38.4.9 nvecs

```
nvecs = cell.empty("")
```

Default: cell.empty("")

Definition at line 91 of file amifun.m.

10.38.4.10 sensiflag

```
sensiflag = logical.empty("")
```

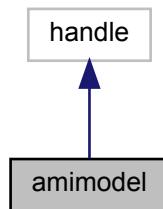
Default: logical.empty("")

Definition at line 99 of file amifun.m.

10.39 amimodel Class Reference

AMIMODEL carries all model definitions including functions and events.

Inheritance diagram for amimodel:



Public Member Functions

- **amimodel** (::string symfun,:string modelName)

amimodel initializes the model object based on the provided symfun and modelName
- noret::substitute **updateRHS** (matlabtypesubstitute xdot)

updateRHS updates the private fun property .fun.xdot.sym (right hand side of the differential equation)
- noret::substitute **update modelName** (matlabtypesubstitute modelName)

updatemodelName updates the modelName
- noret::substitute **updateWrapPath** (matlabtypesubstitute wrap_path)

updatemodelName updates the modelName
- noret::substitute **parseModel** ()

parseModel parses the model definition and computes all necessary symbolic expressions.

- noret::substitute `generateC ()`
`generateC` generates the *c* files which will be used in the compilation.
- noret::substitute `compileC ()`
`compileC` compiles the mex simulation file
- noret::substitute `generateM (::amimodel amimodelo2)`
`generateM` generates the matlab wrapper for the compiled C files.
- noret::substitute `getFun (::struct HTable,::string funstr)`
`getFun` generates symbolic expressions for the requested function.
- noret::substitute `makeEvents ()`
`makeEvents` extracts discontinuities from the model right hand side and converts them into events
- noret::substitute `makeSyms ()`
`makeSyms` extracts symbolic definition from the user provided model and checks them for consistency
- mlhsInnerSubst< matlabypesubstitute > `checkDeps (::struct HTable,::cell deps)`
`checkDeps` checks the dependencies of functions and populates sym fields if necessary
- mlhsInnerSubst< matlabypesubstitute > `loadOldHashes ()`
`loadOldHashes` loads information from a previous compilation of the model.
- mlhsInnerSubst< matlabypesubstitute > `augmento2 ()`
`augmento2` augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward approach later on.
- mlhsInnerSubst< matlabypesubstitute > `augmento2vec ()`
`augmento2vec` augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward approach later on.

Static Public Member Functions

- static noret::substitute `compileAndLinkModel (matlabypesubstitute modelName, matlabypesubstitute modelSourceFolder, matlabypesubstitute coptim, matlabypesubstitute debug, matlabypesubstitute funs, matlabypesubstitute cfun)`
`compileAndLinkModel` compiles the mex simulation file. It does not check if the model files have changed since generating C++ code or whether all files are still present. Use only if you know what you are doing. The safer alternative is rerunning `amiwrap()`.
- static noret::substitute `generateMatlabWrapper (matlabypesubstitute nx, matlabypesubstitute ny, matlabypesubstitute np, matlabypesubstitute nk, matlabypesubstitute nz, matlabypesubstitute o2flag,::amimodel amimodelo2, matlabypesubstitute wrapperFilename, matlabypesubstitute modelName, matlabypesubstitute pscale, matlabypesubstitute forward, matlabypesubstitute adjoint)`
`generateMatlabWrapper` generates the matlab wrapper for the compiled C files.

Public Attributes

- ::struct `sym` = struct.empty("")
symbolic definition struct
- ::struct `fun` = struct.empty("")
struct which stores information for which functions c code needs to be generated
- ::amievent `event` = amievent.empty("")
struct which stores information for which functions c code needs to be generated
- ::string `modelName` = char.empty("")
name of the model
- ::struct `HTable` = struct.empty("")
struct that contains hash values for the symbolic model definitions
- ::bool `debug` = false
flag indicating whether debugging symbols should be compiled

- ::bool **adjoint** = true
flag indicating whether adjoint sensitivities should be enabled
- ::bool **forward** = true
flag indicating whether forward sensitivities should be enabled
- ::double **t0** = 0
default initial time
- ::string **wtype** = char.empty("")
type of wrapper (cvodes/idas)
- ::int **nx** = double.empty("")
number of states
- ::int **nxtrue** = double.empty("")
number of original states for second order sensitivities
- ::int **ny** = double.empty("")
number of observables
- ::int **nytrue** = double.empty("")
number of original observables for second order sensitivities
- ::int **np** = double.empty("")
number of parameters
- ::int **nk** = double.empty("")
number of constants
- ::int **ng** = double.empty("")
number of objective functions
- ::int **nevent** = double.empty("")
number of events
- ::int **nz** = double.empty("")
number of event outputs
- ::int **nztrue** = double.empty("")
number of original event outputs for second order sensitivities
- ::*int **id** = double.empty("")
flag for DAEs
- ::int **ubw** = double.empty("")
upper Jacobian bandwidth
- ::int **lbw** = double.empty("")
lower Jacobian bandwidth
- ::int **n nz** = double.empty("")
number of nonzero entries in Jacobian
- ::*int **sparseidx** = double.empty("")
dataindexes of sparse Jacobian
- ::*int **rowvals** = double.empty("")
rowindexes of sparse Jacobian
- ::*int **colptrs** = double.empty("")
columnindexes of sparse Jacobian
- ::*int **sparseidxB** = double.empty("")
dataindexes of sparse Jacobian
- ::*int **rowvalsB** = double.empty("")
rowindexes of sparse Jacobian
- ::*int **colptrsB** = double.empty("")
columnindexes of sparse Jacobian
- ::*cell **funs** = cell.empty("")
cell array of functions to be compiled
- ::*cell **mfuns** = cell.empty("")

- **coptim** = "-O3"
optimisation flag for compilation
- **param** = "lin"
default parametrisation
- matlabypesubstitute **wrap_path** = char.empty("")
path to wrapper
- matlabypesubstitute **recompile** = false
flag to enforce recompilation of the model
- matlabypesubstitute **cfun** = struct.empty("")
storage for flags determining recompilation of individual functions
- matlabypesubstitute **o2flag** = 0
flag which identifies augmented models 0 indicates no augmentation 1 indicates augmentation by first order sensitivities (yields second order sensitivities) 2 indicates augmentation by one linear combination of first order sensitivities (yields hessian-vector product)
- matlabypesubstitute **z2event** = double.empty("")
vector that maps outputs to events
- matlabypesubstitute **splineflag** = false
flag indicating whether the model contains spline functions
- matlabypesubstitute **minflag** = false
flag indicating whether the model contains min functions
- matlabypesubstitute **maxflag** = false
flag indicating whether the model contains max functions
- ::int **nw** = 0
number of derived variables w, w is used for code optimization to reduce the number of frequently occurring expressions
- ::int **ndwdx** = 0
number of derivatives of derived variables w, dwdx
- ::int **ndwdp** = 0
number of derivatives of derived variables w, dwdp

10.39.1 Detailed Description

Definition at line 17 of file amimodel.m.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 amimodel()

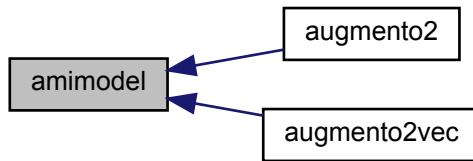
```
amimodel (
    ::string symfun,
    ::string modelname )
```

Parameters

symfun	this is the string to the function which generates the modelstruct. You can also directly pass the struct here
modelname	name of the model

Definition at line 515 of file amimodel.m.

Here is the caller graph for this function:



10.39.3 Member Function Documentation

10.39.3.1 updateRHS()

```
noret::substitute updateRHS (
    matlabtypesubstitute xdot )
```

Parameters

<i>xdot</i>	new right hand side of the differential equation
-------------	--

Return values

<i>xdot</i>	void
-------------	------

Definition at line 612 of file amimodel.m.

10.39.3.2 updatemodelName()

```
noret::substitute updatemodelName (
    matlabtypesubstitute modelname )
```

Parameters

<i>modelname</i>	new modelname
------------------	---------------

Return values

<i>modelname</i>	void
------------------	------

Definition at line 627 of file amimodel.m.

10.39.3.3 updateWrapPath()

```
noret::substitute updateWrapPath (
    matlabtypesubstitute wrap_path )
```

Parameters

<i>wrap_path</i>	new wrap_path
------------------	---------------

Return values

<i>wrap_path</i>	void
------------------	------

Definition at line 640 of file amimodel.m.

10.39.3.4 parseModel()

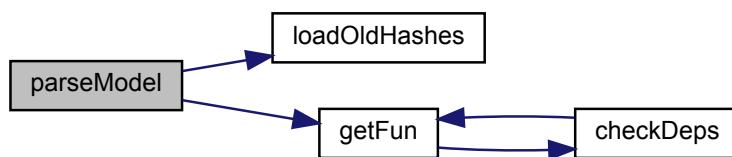
```
noret::substitute parseModel ( )
```

Return values

<i>void</i>	
-------------	--

Definition at line 18 of file parseModel.m.

Here is the call graph for this function:



10.39.3.5 generateC()

```
noret::substitute generateC ( )
```

Return values

<i>void</i>	
-------------	--

Definition at line 18 of file generateC.m.

10.39.3.6 compileC()

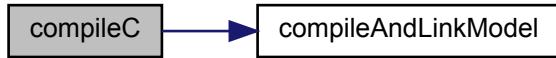
```
noret::substitute compileC ( )
```

Return values

<i>void</i>	
-------------	--

Definition at line 18 of file compileC.m.

Here is the call graph for this function:

**10.39.3.7 generateM()**

```
noret::substitute generateM ( ::amimodel amimodelo2 )
```

Parameters

<i>amimodelo2</i>	this struct must contain all necessary symbolic definitions for second order sensitivities
-------------------	--

Return values

<i>amimodelo2</i>	<i>void</i>
-------------------	-------------

Definition at line 18 of file generateM.m.

Here is the call graph for this function:



10.39.3.8 getFun()

```
noret::substitute getFun (   
    ::struct HTable,  
    ::string funstr )
```

Parameters

<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation
<i>funstr</i>	function for which symbolic expressions should be computed

Return values

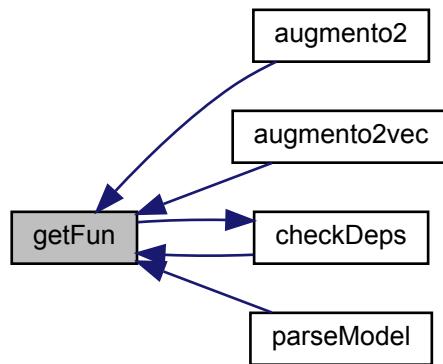
<i>funstr</i>	void
---------------	------

Definition at line 18 of file getFun.m.

Here is the call graph for this function:



Here is the caller graph for this function:



10.39.3.9 makeEvents()

```
noret::substitute makeEvents ( )
```

Return values

<code>void</code>	
-------------------	--

Definition at line 18 of file `makeEvents.m`.

Here is the call graph for this function:



10.39.3.10 makeSyms()

```
noret::substitute makeSyms ( )
```

Return values

<i>void</i>	
-------------	--

Definition at line 18 of file makeSyms.m.

10.39.3.11 checkDeps()

```
mlhsInnerSubst<::bool > checkDeps (
    ::struct HTable,
    ::cell deps )
```

Parameters

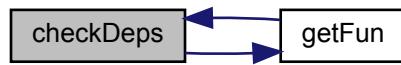
<i>HTable</i>	struct with reference hashes of functions in its fields
<i>deps</i>	cell array with containing a list of dependencies

Return values

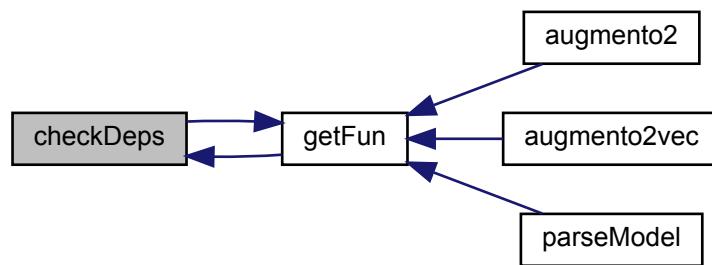
<i>cflag</i>	boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable
--------------	---

Definition at line 18 of file checkDeps.m.

Here is the call graph for this function:



Here is the caller graph for this function:



10.39.3.12 loadOldHashes()

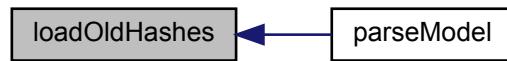
```
mlhsInnerSubst<::struct > loadOldHashes ( )
```

Return values

<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation
---------------	---

Definition at line 18 of file loadOldHashes.m.

Here is the caller graph for this function:



10.39.3.13 augmento2()

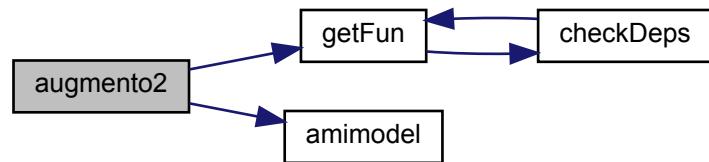
```
mlhsInnerSubst< matlabtypesubstitute > augmento2 ( )
```

Return values

<i>this</i>	augmented system which contains symbolic definition of the original system and its sensitivities
-------------	--

Definition at line 18 of file augmento2.m.

Here is the call graph for this function:



10.39.3.14 augmento2vec()

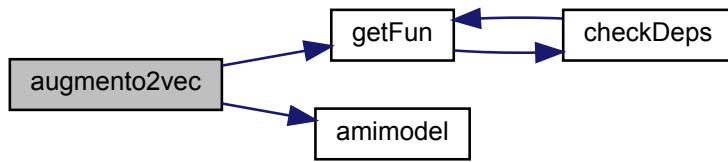
```
mlhsInnerSubst<::amimodel > augmento2vec ( )
```

Return values

<i>modelo2vec</i>	augmented system which contains symbolic definition of the original system and its sensitivities
-------------------	--

Definition at line 18 of file augmento2vec.m.

Here is the call graph for this function:



10.39.3.15 compileAndLinkModel()

```
noret::substitute compileAndLinkModel (
    matlabtypesubstitute modelName,
    matlabtypesubstitute modelSourceFolder,
    matlabtypesubstitute coptim,
    matlabtypesubstitute debug,
    matlabtypesubstitute funs,
    matlabtypesubstitute cfun ) [static]
```

Parameters

<i>modelName</i>	name of the model as specified for amiwrap()
<i>modelSourceFolder</i>	path to model source directory
<i>coptim</i>	optimization flags
<i>debug</i>	enable debugging
<i>funs</i>	array with names of the model functions, will be guessed from source files if left empty
<i>cfun</i>	struct indicating which files should be recompiled

Return values

<i>cfun</i>	void
-------------	------

Definition at line 18 of file compileAndLinkModel.m.

Here is the caller graph for this function:



10.39.3.16 generateMatlabWrapper()

```

noret::substitute generateMatlabWrapper (
    matlabtypesubstitute nx,
    matlabtypesubstitute ny,
    matlabtypesubstitute np,
    matlabtypesubstitute nk,
    matlabtypesubstitute nz,
    matlabtypesubstitute o2flag,
    ::amimodel amimodelo2,
    matlabtypesubstitute wrapperFilename,
    matlabtypesubstitute modelname,
    matlabtypesubstitute pscale,
    matlabtypesubstitute forward,
    matlabtypesubstitute adjoint ) [static]

```

Parameters

<i>nx</i>	number of states
<i>ny</i>	number of observables
<i>np</i>	number of parameters
<i>nk</i>	number of fixed parameters
<i>nz</i>	number of events
<i>o2flag</i>	<i>o2flag</i>
<i>amimodelo2</i>	this struct must contain all necessary symbolic definitions for second order sensitivities
<i>wrapperFilename</i>	output filename
<i>modelname</i>	name of the model
<i>pscale</i>	default parameter scaling
<i>forward</i>	has forward sensitivity equations
<i>adjoint</i>	has adjoint sensitivity equations

Return values

<i>adjoint</i>	void
----------------	------

Definition at line 18 of file `generateMatlabWrapper.m`.

Here is the caller graph for this function:



10.39.4 Member Data Documentation

10.39.4.1 sym

```
sym = struct.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 27 of file amimodel.m.

10.39.4.2 fun

```
fun = struct.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 38 of file amimodel.m.

10.39.4.3 event

```
event = amievent.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: amievent.empty("")

Definition at line 49 of file amimodel.m.

10.39.4.4 modelname

```
modelname = char.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: char.empty("")

Definition at line 61 of file amimodel.m.

10.39.4.5 HTable

```
HTable = struct.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 72 of file amimodel.m.

10.39.4.6 debug

```
debug = false
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: false

Definition at line 83 of file amimodel.m.

10.39.4.7 adjoint

```
adjoint = true
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: true

Definition at line 94 of file amimodel.m.

10.39.4.8 forward

```
forward = true
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: true

Definition at line 105 of file amimodel.m.

10.39.4.9 t0

```
t0 = 0
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: 0

Definition at line 116 of file amimodel.m.

10.39.4.10 wtype

```
wtype = char.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: char.empty("")

Definition at line 127 of file amimodel.m.

10.39.4.11 nx

```
nx = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: double.empty("")

Definition at line 138 of file amimodel.m.

10.39.4.12 nxtrue

```
nxtrue = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: double.empty("")

Definition at line 149 of file amimodel.m.

10.39.4.13 ny

```
ny = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: double.empty("")

Definition at line 160 of file amimodel.m.

10.39.4.14 nytrue

```
nytrue = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: double.empty("")

Definition at line 171 of file amimodel.m.

10.39.4.15 np

```
np = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: double.empty("")

Definition at line 182 of file amimodel.m.

10.39.4.16 nk

```
nk = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 193 of file amimodel.m.

10.39.4.17 ng

```
ng = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 204 of file amimodel.m.

10.39.4.18 nevent

```
nevent = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 215 of file amimodel.m.

10.39.4.19 nz

```
nz = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 226 of file amimodel.m.

10.39.4.20 nztrue

```
nztrue = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 237 of file amimodel.m.

10.39.4.21 id

```
id = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 248 of file amimodel.m.

10.39.4.22 ubw

```
ubw = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 259 of file amimodel.m.

10.39.4.23 lbw

```
lbw = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 270 of file amimodel.m.

10.39.4.24 nnz

```
nnz = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 281 of file amimodel.m.

10.39.4.25 sparseidx

```
sparseidx = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 292 of file amimodel.m.

10.39.4.26 rowvals

```
rowvals = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 303 of file amimodel.m.

10.39.4.27 colptrs

```
colptrs = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

Default: double.empty("")

Definition at line 314 of file amimodel.m.

10.39.4.28 sparseidxB

```
sparseidxB = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 325 of file amimodel.m.

10.39.4.29 rowvalsB

```
rowvalsB = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 336 of file amimodel.m.

10.39.4.30 colptrsB

```
colptrsB = double.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 347 of file amimodel.m.

10.39.4.31 funs

```
funs = cell.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: cell.empty("")

Definition at line 358 of file amimodel.m.

10.39.4.32 mfun

```
mfun = cell.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: cell.empty("")

Definition at line 369 of file amimodel.m.

10.39.4.33 coptim

```
coptim = "-O3"
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: "-O3"

Definition at line 380 of file amimodel.m.

10.39.4.34 param

```
param = "lin"
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: "lin"

Definition at line 391 of file amimodel.m.

10.39.4.35 wrap_path

```
wrap_path = char.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

Matlab documentation of property attributes.

Default: char.empty("")

Definition at line 402 of file amimodel.m.

10.39.4.36 recompile

```
recompile = false
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: false

Definition at line 413 of file amimodel.m.

10.39.4.37 cfun

```
cfun = struct.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: struct.empty("")

Definition at line 424 of file amimodel.m.

10.39.4.38 o2flag

```
o2flag = 0
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: 0

Definition at line 436 of file amimodel.m.

10.39.4.39 z2event

```
z2event = double.empty("")
```

Default: double.empty("")

Definition at line 455 of file amimodel.m.

10.39.4.40 splineflag

```
splineflag = false
```

Default: false

Definition at line 463 of file amimodel.m.

10.39.4.41 minflag

```
minflag = false
```

Default: false

Definition at line 471 of file amimodel.m.

10.39.4.42 maxflag

```
maxflag = false
```

Default: false

Definition at line 479 of file amimodel.m.

10.39.4.43 nw

```
nw = 0
```

Default: 0

Definition at line 487 of file amimodel.m.

10.39.4.44 ndwdx

```
ndwdx = 0
```

Default: 0

Definition at line 496 of file amimodel.m.

10.39.4.45 ndwdp

```
ndwdp = 0
```

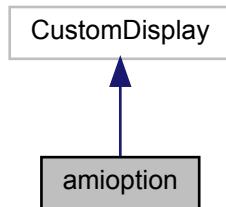
Default: 0

Definition at line 504 of file amimodel.m.

10.40 amioption Class Reference

AMIOPTION provides an option container to pass simulation parameters to the simulation routine.

Inheritance diagram for amioption:



Public Member Functions

- [amioption](#) (matlabtypesubstitute varargin)

amioptions Construct a new *amioptions* object *OPTS* = [amioption\(\)](#) creates a set of options with each option set to its default value.

Public Attributes

- matlabtypesubstitute **atol** = 1e-16
absolute integration tolerance
- matlabtypesubstitute **rtol** = 1e-8
relative integration tolerance
- matlabtypesubstitute **maxsteps** = 1e4
maximum number of integration steps
- matlabtypesubstitute **quad_atol** = 1e-12
absolute quadrature tolerance
- matlabtypesubstitute **quad_rtol** = 1e-8
relative quadrature tolerance
- matlabtypesubstitute **maxstepsB** = 0
maximum number of integration steps
- matlabtypesubstitute **ss_atol** = 1e-16

- `matlabypesubstitute ss_rtol = 1e-8`
absolute steady state tolerace
- `matlabypesubstitute sens_ind = double.empty("")`
relative steady state tolerace
- `matlabypesubstitute tstart = 0`
index of parameters for which the sensitivities are computed
- `matlabypesubstitute lmm = 2`
starting time of the simulation
- `matlabypesubstitute iter = 2`
linear multistep method.
- `matlabypesubstitute linsol = 9`
iteration method for linear multistep.
- `matlabypesubstitute stldet = true`
linear solver
- `matlabypesubstitute interpType = 1`
stability detection flag
- `matlabypesubstitute ism = 1`
interpolation type
- `matlabypesubstitute nmaxevent = 10`
forward sensitivity mode
- `matlabypesubstitute sensi_meth = 1`
sensitivity method
- `matlabypesubstitute sensi = 0`
sensitivity order
- `matlabypesubstitute ordering = 0`
number of reported events
- `matlabypesubstitute ss = 0`
reordering of states
- `matlabypesubstitute x0 = double.empty("")`
steady state sensitivity flag
- `matlabypesubstitute sx0 = double.empty("")`
custom initial state
- `matlabypesubstitute newton_maxsteps = 40`
custom initial sensitivity
- `matlabypesubstitute newton_maxlinsteps = 100`
newton solver: maximum newton steps
- `matlabypesubstitute newton_preq = false`
newton solver: maximum linear steps
- `matlabypesubstitute z2event = double.empty("")`
preequilibration of system via newton solver
- `matlabypesubstitute pscale = "[]"`
mapping of event ouputs to events
- `matlabypesubstitute parameter scaling Single value or vector matching sens_ind. Valid options are "log", "log10" and "lin" for log, log10 or unscaled parameters p. Use [] for default as specified in the model (fallback: lin).`
parameter scaling Single value or vector matching sens_ind. Valid options are "log", "log10" and "lin" for log, log10 or unscaled parameters p. Use [] for default as specified in the model (fallback: lin).

10.40.1 Detailed Description

Definition at line 17 of file amioption.m.

10.40.2 Constructor & Destructor Documentation

10.40.2.1 amioption()

```
amioption (
    matlabtypesubstitute varargin )
```

OPTS = amioption(PARAM, VAL, ...) creates a set of options with the named parameters altered with the specified values.

OPTS = amioption(OLDOPTS, PARAM, VAL, ...) creates a copy of OLDOPTS with the named parameters altered with the specified value

Note: to see the parameters, check the documentation page for amioption

Parameters

<code>varargin</code>	input to construct amioption object, see function function description
-----------------------	--

Definition at line 259 of file amioption.m.

10.40.3 Member Data Documentation

10.40.3.1 atol

```
atol = 1e-16
```

Default: 1e-16

Definition at line 28 of file amioption.m.

10.40.3.2 rtol

```
rtol = 1e-8
```

Default: 1e-8

Definition at line 36 of file amioption.m.

10.40.3.3 maxsteps

```
maxsteps = 1e4
```

Default: 1e4

Definition at line 44 of file amioption.m.

10.40.3.4 quad_atol

```
quad_atol = 1e-12
```

Default: 1e-12

Definition at line 52 of file amioption.m.

10.40.3.5 quad_rtol

```
quad_rtol = 1e-8
```

Default: 1e-8

Definition at line 60 of file amioption.m.

10.40.3.6 maxstepsB

```
maxstepsB = 0
```

Default: 0

Definition at line 68 of file amioption.m.

10.40.3.7 ss_atol

```
ss_atol = 1e-16
```

Default: 1e-16

Definition at line 76 of file amioption.m.

10.40.3.8 ss_rtol

```
ss_rtol = 1e-8
```

Default: 1e-8

Definition at line 84 of file amioption.m.

10.40.3.9 sens_ind

```
sens_ind = double.empty("")
```

Default: double.empty("")

Definition at line 92 of file amioption.m.

10.40.3.10 tstart

```
tstart = 0
```

Default: 0

Definition at line 100 of file amioption.m.

10.40.3.11 lmm

```
lmm = 2
```

Default: 2

Definition at line 108 of file amioption.m.

10.40.3.12 iter

```
iter = 2
```

Default: 2

Definition at line 116 of file amioption.m.

10.40.3.13 linsol

```
linsol = 9
```

Default: 9

Definition at line 124 of file amioption.m.

10.40.3.14 stldet

```
stldet = true
```

Default: true

Definition at line 132 of file amioption.m.

10.40.3.15 interpType

```
interpType = 1
```

Default: 1

Definition at line 140 of file amioption.m.

10.40.3.16 ism

```
ism = 1
```

Default: 1

Definition at line 148 of file amioption.m.

10.40.3.17 sensi_meth

```
sensi_meth = 1
```

Default: 1

Note

This property has custom functionality when its value is changed.

Definition at line 156 of file amioption.m.

10.40.3.18 sensi

```
sensi = 0
```

Default: 0**Note**

This property has custom functionality when its value is changed.

Definition at line 164 of file amioption.m.

10.40.3.19 nmaxevent

```
nmaxevent = 10
```

Default: 10

Definition at line 172 of file amioption.m.

10.40.3.20 ordering

```
ordering = 0
```

Default: 0

Definition at line 180 of file amioption.m.

10.40.3.21 ss

```
ss = 0
```

Default: 0

Definition at line 188 of file amioption.m.

10.40.3.22 x0

```
x0 = double.empty("")
```

Default: double.empty("")

Definition at line 196 of file amioption.m.

10.40.3.23 sx0

```
sx0 = double.empty("")
```

Default: double.empty("")

Definition at line 204 of file amioption.m.

10.40.3.24 newton_maxsteps

```
newton_maxsteps = 40
```

Default: 40

Note

This property has custom functionality when its value is changed.

Definition at line 212 of file amioption.m.

10.40.3.25 newton_maxlinsteps

```
newton_maxlinsteps = 100
```

Default: 100

Note

This property has custom functionality when its value is changed.

Definition at line 220 of file amioption.m.

10.40.3.26 newton_preq

```
newton_preq = false
```

Default: false

Note

This property has custom functionality when its value is changed.

Definition at line 228 of file amioption.m.

10.40.3.27 z2event

```
z2event = double.empty("")
```

Default: double.empty("")

Definition at line 236 of file amioption.m.

10.40.3.28 pscale

```
pscale = "[ ]"
```

Default: "[]"

Note

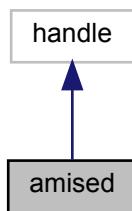
This property has custom functionality when its value is changed.

Definition at line 244 of file amioption.m.

10.41 amised Class Reference

AMISED is a container for SED-ML objects.

Inheritance diagram for amised:



Public Member Functions

- [amised](#) (matlabtypesubstitute sedname)

amised reads in an SEDML document using the JAVA binding of libSEDML

Public Attributes

- matlabypesubstitute `model` = struct("event",[],'sym',[])

amimodel from the specified model
- matlabypesubstitute `modelname` = {""}

cell array of model identifiers
- matlabypesubstitute `sedml` = struct.empty("")

stores the struct tree from the xml definition
- matlabypesubstitute `outputcount` = "[]"

count the number of outputs per model
- matlabypesubstitute `varidx` = "[]"

indexes for dataGenerators
- matlabypesubstitute `varsym` = sym("[]")

symbolic expressions for variables
- matlabypesubstitute `datasym` = sym("[]")

symbolic expressions for data

10.41.1 Detailed Description

Definition at line 17 of file amised.m.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 amised()

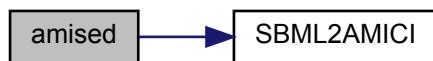
```
amised (
    matlabypesubstitute sedname )
```

Parameters

<code>sedname</code>	name/path of the SEDML document
----------------------	---------------------------------

Definition at line 112 of file amised.m.

Here is the call graph for this function:



10.41.3 Member Data Documentation

10.41.3.1 model

```
model = struct("'event',[],'sym',[ ])
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: struct("event",[],'sym',[])

Definition at line 27 of file amised.m.

10.41.3.2 modelname

```
modelname = { "" }
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: {""}

Definition at line 38 of file amised.m.

10.41.3.3 sedml

```
sedml = struct.empty("")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 49 of file amised.m.

10.41.3.4 outputcount

```
outputcount = " [ ] "
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
Matlab documentation of property attributes.
Default: "[]"

Definition at line 60 of file amised.m.

10.41.3.5 varidx

```
varidx = "[ ]"
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: "[]"

Definition at line 71 of file amised.m.

10.41.3.6 varsym

```
varsym = sym(" [ ] ")
```

Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: sym("[])")

Definition at line 82 of file amised.m.

10.41.3.7 datasym

```
datasym = sym(" [ ] ")
```

Note

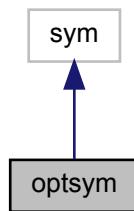
This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public
[Matlab documentation of property attributes.](#)
Default: sym("[])")

Definition at line 93 of file amised.m.

10.42 optsym Class Reference

OPTSYM is an auxiliary class to gain access to the private symbolic property `s` which is necessary to be able to call `symobj::optimize` on it.

Inheritance diagram for optsym:



Public Member Functions

- `optsym (::sym symbol)`
optsym converts the symbolic object into a optsym object
- `mlhsInnerSubst<::sym > getoptimized ()`
getoptimized calls symobj::optimize on the optsym object

10.42.1 Detailed Description

Definition at line 17 of file optsym.m.

10.42.2 Constructor & Destructor Documentation

10.42.2.1 optsym()

```
optsym (
    ::sym symbol )
```

Parameters

<code>symbol</code>	symbolic object
---------------------	-----------------

Definition at line 32 of file optsym.m.

10.42.3 Member Function Documentation

10.42.3.1 getoptimized()

```
mlhsInnerSubst<::sym > getoptimized ( )
```

Return values

<code>out</code>	optimized symbolic object
------------------	---------------------------

Definition at line 42 of file optsym.m.

11 File Documentation

11.1 am_and.m File Reference

am_and is the amici implementation of the symbolic and function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_and** (:sym a,:sym b)
am_and is the amici implementation of the symbolic and function

11.1.1 Function Documentation

11.1.1.1 am_and()

```
mlhsInnerSubst< matlabtypesubstitute > am_and (
    ::sym a,
    ::sym b )
```

Parameters

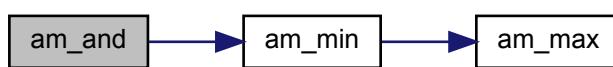
<i>a</i>	first input parameter
<i>b</i>	second input parameter

Return values

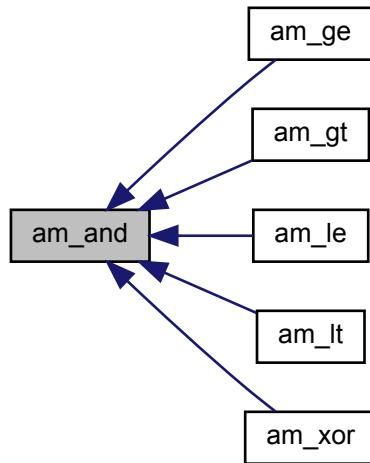
<i>fun</i>	logical value, negative for false, positive for true
------------	--

Definition at line 17 of file am_and.m.

Here is the call graph for this function:



Here is the caller graph for this function:



11.2 am_eq.m File Reference

am_eq is currently a placeholder that simply produces an error message

Functions

- mlhsInnerSubst< matlabtypesubstitute > [am_eq](#) (matlabtypesubstitute varargin)
am_eq is currently a placeholder that simply produces an error message

11.2.1 Function Documentation

11.2.1.1 am_eq()

```
mlhsInnerSubst< matlabtypesubstitute > am_eq (   
 matlabtypesubstitute varargin )
```

Parameters

<code>varargin</code>	elements for chain of equalities
-----------------------	----------------------------------

Return values

<code>fun</code>	logical value, negative for false, positive for true
------------------	--

Definition at line 17 of file am_eq.m.

11.3 am_ge.m File Reference

am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_ge** (:sym varargin)

am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

11.3.1 Function Documentation

11.3.1.1 am_ge()

```
mlhsInnerSubst< matlabtypesubstitute > am_ge (
    ::sym varargin )
```

Parameters

<i>varargin</i>	chain of input parameters
-----------------	---------------------------

Return values

<i>fun</i>	a >= b logical value, negative for false, positive for true
------------	---

Definition at line 17 of file am_ge.m.

Here is the call graph for this function:



11.4 am_gt.m File Reference

am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_gt** (:sym varargin)

am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

11.4.1 Function Documentation

11.4.1.1 am_gt()

```
mlhsInnerSubst< matlabtypesubstitute > am_gt (
    ::sym varargin )
```

Parameters

varargin	chain of input parameters
----------	---------------------------

Return values

fun	a > b logical value, negative for false, positive for true
-----	--

Definition at line 17 of file am_gt.m.

Here is the call graph for this function:



11.5 am_if.m File Reference

am_if is the amici implementation of the symbolic if function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_if** (:sym condition,:sym truepart,:sym falsepart)

am_if is the amici implementation of the symbolic if function

11.5.1 Function Documentation

11.5.1.1 am_if()

```
mlhsInnerSubst< matlabtypesubstitute > am_if (
    ::sym condition,
    ::sym truepart,
    ::sym falsepart )
```

Parameters

<i>condition</i>	logical value
<i>truepart</i>	value if condition is true
<i>falsepart</i>	value if condition is false

Return values

<i>fun</i>	if condition is true truepart, else falsepart
------------	---

Definition at line 17 of file am_if.m.

Here is the caller graph for this function:



11.6 am_le.m File Reference

am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > [am_le](#) (::sym varargin)

am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

11.6.1 Function Documentation

11.6.1.1 am_le()

```
mlhsInnerSubst< matlabtypesubstitute > am_le (
    ::sym varargin )
```

Parameters

<i>varargin</i>	chain of input parameters
-----------------	---------------------------

Return values

<i>fun</i>	$a <= b$ logical value, negative for false, positive for true
------------	---

Definition at line 17 of file am_le.m.

Here is the call graph for this function:

**11.7 am_lt.m File Reference**

am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_lt** (:sym varargin)

am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

11.7.1 Function Documentation**11.7.1.1 am_lt()**

```
mlhsInnerSubst< matlabtypesubstitute > am_lt (
    ::sym varargin )
```

Parameters

<i>varargin</i>	chain of input parameters
-----------------	---------------------------

Return values

<i>fun</i>	$a < b$ logical value, negative for false, positive for true
------------	--

Definition at line 17 of file am_lt.m.

Here is the call graph for this function:



11.8 am_max.m File Reference

am_max is the amici implementation of the symbolic max function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_max** (:sym a,:sym b)
am_max is the amici implementation of the symbolic max function

11.8.1 Function Documentation

11.8.1.1 am_max()

```
mlhsInnerSubst< matlabtypesubstitute > am_max (
    ::sym a,
    ::sym b )
```

Parameters

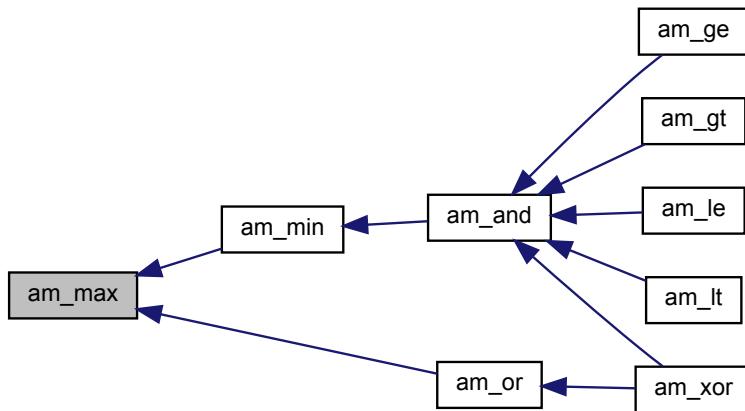
<i>a</i>	first input parameter
<i>b</i>	second input parameter

Return values

<i>fun</i>	maximum of a and b
------------	--------------------

Definition at line 17 of file am_max.m.

Here is the caller graph for this function:



11.9 am_min.m File Reference

`am_min` is the amici implementation of the symbolic min function

Functions

- mlhsInnerSubst< matlabtypesubstitute > `am_min` (:sym a,:sym b)
am_min is the amici implementation of the symbolic min function

11.9.1 Function Documentation

11.9.1.1 am_min()

```
mlhsInnerSubst< matlabtypesubstitute > am_min (
    ::sym a,
    ::sym b )
```

Parameters

<code>a</code>	first input parameter
<code>b</code>	second input parameter

Return values

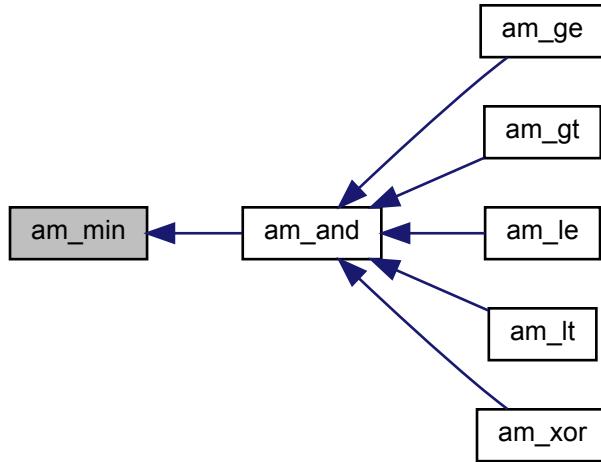
<code>fun</code>	minimum of a and b
------------------	--------------------

Definition at line 17 of file am_min.m.

Here is the call graph for this function:



Here is the caller graph for this function:



11.10 am_or.m File Reference

am_or is the amici implementation of the symbolic or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_or** (:sym a,:sym b)
am_or is the amici implementation of the symbolic or function

11.10.1 Function Documentation

11.10.1.1 am_or()

```

mlhsInnerSubst< matlabtypesubstitute > am_or (
    ::sym a,
    ::sym b )
  
```

Parameters

<i>a</i>	first input parameter
<i>b</i>	second input parameter

Return values

<i>fun</i>	logical value, negative for false, positive for true
------------	--

Definition at line 17 of file am_or.m.

Here is the call graph for this function:



Here is the caller graph for this function:



11.11 am_piecewise.m File Reference

am_piecewise is the amici implementation of the mathml piecewise function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_piecewise** (matlabtypesubstitute *piece*, matlabtypesubstitute *condition*, matlabtypesubstitute *default*)

am_piecewise is the amici implementation of the mathml piecewise function

11.11.1 Function Documentation

11.11.1.1 am_piecewise()

```

mlhsInnerSubst< matlabtypesubstitute > am_piecewise (
    matlabtypesubstitute piece,
    matlabtypesubstitute condition,
    matlabtypesubstitute default )
  
```

Parameters

<i>piece</i>	value if condition is true
<i>condition</i>	logical value
<i>default</i>	value if condition is false

Return values

<i>fun</i>	return value, piece if condition is true, default if not
------------	--

Definition at line 17 of file am_piecewise.m.

Here is the call graph for this function:



11.12 am_stepfun.m File Reference

am_stepfun is the amici implementation of the step function

Functions

- mlhsInnerSubst< matlabtypesubstitute > [am_stepfun](#) (:sym t, matlabtypesubstitute tstart, matlabtypesubstitute vstart, matlabtypesubstitute tend, matlabtypesubstitute vend)

am_stepfun is the amici implementation of the step function

11.12.1 Function Documentation

11.12.1.1 am_stepfun()

```
mlhsInnerSubst< matlabtypesubstitute > am_stepfun (
    ::sym t,
    matlabtypesubstitute tstart,
    matlabtypesubstitute vstart,
    matlabtypesubstitute tend,
    matlabtypesubstitute vend )
```

Parameters

<i>t</i>	input variable
<i>tstart</i>	input variable value at which the step starts
<i>vstart</i>	value during the step
<i>tend</i>	input variable value at which the step end
<i>vend</i>	value after the step

Return values

<i>fun</i>	0 before tstart, vstart between tstart and tend and vend after tend
------------	---

Definition at line 17 of file am_stepfun.m.

11.13 am_xor.m File Reference

am_xor is the amici implementation of the symbolic exclusive or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > **am_xor** (:sym a,:sym b)
am_xor is the amici implementation of the symbolic exclusive or function

11.13.1 Function Documentation

11.13.1.1 am_xor()

```
mlhsInnerSubst< matlabtypesubstitute > am_xor (
    ::sym a,
    ::sym b )
```

Parameters

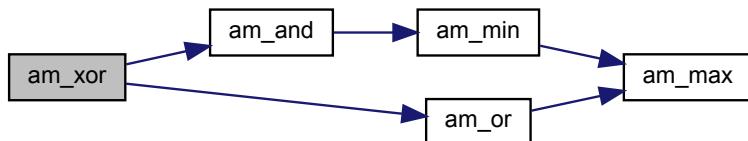
<i>a</i>	first input parameter
<i>b</i>	second input parameter

Return values

<i>fun</i>	logical value, negative for false, positive for true
------------	--

Definition at line 17 of file am_xor.m.

Here is the call graph for this function:



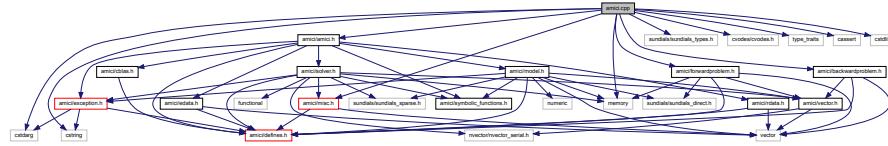
11.14 amici.cpp File Reference

core routines for integration

```
#include "amici/amici.h"
#include "amici/backwardproblem.h"
#include "amici/forwardproblem.h"
#include "amici/misc.h"
#include <sundials/sundials_types.h>
#include <cvodes/cvodes.h>
#include <type_traits>
#include <cassert>
#include <cstdlib>
#include <cstring>
#include <cstdarg>
#include <memory>
```

Include dependency graph for amici.cpp:

Include dependency graph for amici.cpp:



Namespaces

- amici

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Functions

- std::unique_ptr<ReturnData> runAmiciSimulation (Solver &solver, const ExpData *edata, Model &model)
 - void printErrMsgIdAndTxt (const char *identifier, const char *format,...)
 - void printWarnMsgIdAndTxt (const char *identifier, const char *format,...)
 - std::vector< std::unique_ptr< ReturnData > > runAmiciSimulations (Solver const &solver, const std::vector< ExpData * > &edatas, Model const &model, int num_threads)

11.15 amiwrap.m File Reference

AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

Functions

- `noret::substitute` [amiwrap](#) (`matlabtypesubstitute` `varargin`)

AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

11.15.1 Function Documentation

11.15.1.1 amiwrap()

```
noret:::substitute amiwrap (
    matlabtypesubstitute varargin )
```

Parameters**varargin**

```
amiwrap ( modelname, symfun, tdir, o2flag )
```

Required Parameters for varargin:

- modelname specifies the name of the model which will be later used for the naming of the simulation file
- symfun specifies a function which executes model definition see [MATLAB Interface](#) for details
- tdir target directory where the simulation file should be placed **Default:** \$AMICIDIR/models/modelname
- o2flag boolean whether second order sensitivities should be enabled **Default:** false

Return values

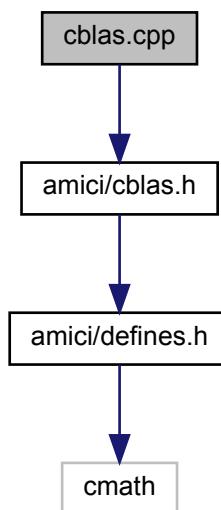
<i>o2flag</i>	void
---------------	------

Definition at line 17 of file amiwrap.m.

11.16 cblas.cpp File Reference

BLAS routines required by AMICI.

```
#include "amici/cblas.h"
Include dependency graph for cblas.cpp:
```



Namespaces

- **amici**

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Functions

- void **amici_dgemm** (BLASLayout layout, BLASTranspose TransA, BLASTranspose 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** (BLASLayout layout, BLASTranspose 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)
- void **amici_daxpy** (int n, double alpha, const double *x, const int incx, double *y, int incy)

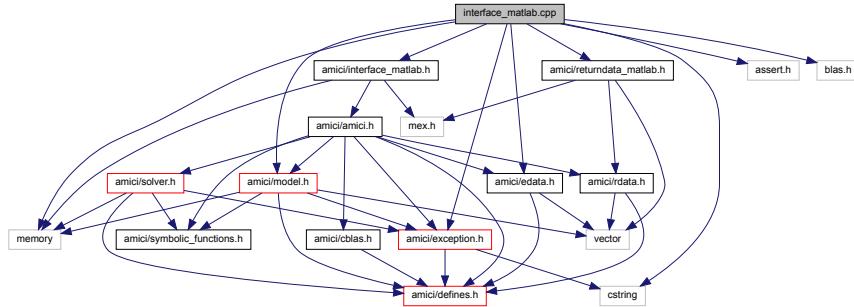
*Compute $y = a*x + y$.*

11.17 interface_matlab.cpp File Reference

core routines for mex interface

```
#include "amici/interface_matlab.h"
#include "amici/model.h"
#include "amici/exception.h"
#include "amici/edata.h"
#include "amici/returndata_matlab.h"
#include <assert.h>
#include <blas.h>
#include <cstring>
#include <memory>
```

Include dependency graph for interface_matlab.cpp:



Namespaces

- **amici**

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Enumerations

- enum `mexRhsArguments` {
 `RHS_TIMEPOINTS`, `RHS_PARAMETERS`, `RHS_CONSTANTS`, `RHS_OPTIONS`,
 `RHS_PLIST`, `RHS_XSCALE_UNUSED`, `RHS_INITIALIZATION`, `RHS_DATA`,
 `RHS_NUMARGS_REQUIRED` = `RHS_DATA`, `RHS_NUMARGS` }

The mexFunctionArguments enum takes care of the ordering of mex file arguments (indexing in prhs)

Functions

- int `dbl2int` (const double x)
- char `amici_blasCBlasTransToBlasTrans` (BLASTranspose trans)
- void `amici_dgemm` (BLASLayout layout, BLASTranspose TransA, BLASTranspose 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` (BLASLayout layout, BLASTranspose 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)
- void `amici_daxpy` (int n, double alpha, const double *x, const int incx, double *y, int incy)

*Compute $y = a*x + y$.*
- std::vector< realtype > `mxArrayToVector` (const mxArray *array, int length)
- std::unique_ptr< ExpData > `expDataFromMatlabCall` (const mxArray *prhs[], const Model &model)
- void `setSolverOptions` (const mxArray *prhs[], int nrhs, Solver &solver)

setSolverOptions solver options from the matlab call to a solver object
- void `setModelData` (const mxArray *prhs[], int nrhs, Model &model)

setModelData sets data from the matlab call to the model object
- void `mexFunction` (int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])

11.17.1 Detailed Description

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

11.17.2 Function Documentation

11.17.2.1 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

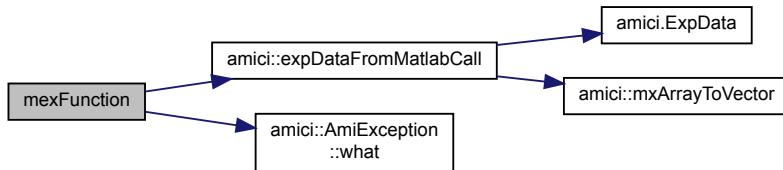
<code>nlhs</code>	number of output arguments of the matlab call
<code>plhs</code>	pointer to the array of output arguments
<code>nrhs</code>	number of input arguments of the matlab call
<code>prhs</code>	pointer to the array of input arguments

Returns

<code>void</code>

Definition at line 525 of file interface_matlab.cpp.

Here is the call graph for this function:



11.18 SBML2AMICI.m File Reference

SBML2AMICI generates AMICI model definition files from SBML.

Functions

- `noret::substitute SBML2AMICI (matlabtypesubstitute filename, matlabtypesubstitute modelname)`
SBML2AMICI generates AMICI model definition files from SBML.

11.18.1 Function Documentation

11.18.1.1 SBML2AMICI()

```
noret::substitute SBML2AMICI (
    matlabtypesubstitute filename,
    matlabtypesubstitute modelname )
```

Parameters

<code>filename</code>	name of the SBML file (withouth extension)
<code>modelname</code>	name of the model, this will define the name of the output file (default: input filename)

Return values

<code>modelname</code>	<code>void</code>
------------------------	-------------------

Definition at line 17 of file SBML2AMICI.m.

Here is the caller graph for this function:



11.19 spline.cpp File Reference

definition of spline functions

Namespaces

- [amici](#)

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

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[])
Evaluate the cubic spline function.
- double [sinteg](#) (int n, double u, double x[], double y[], double b[], double c[], double d[])

11.19.1 Detailed Description

Author

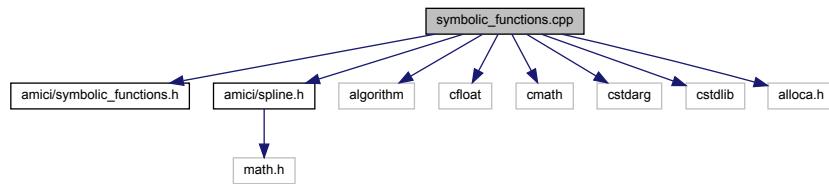
Peter & Nigel, Design Software, 42 Gubberley St, Kenmore, 4069, Australia.

11.20 symbolic_functions.cpp File Reference

definition of symbolic functions

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

```
#include <alloca.h>
Include dependency graph for symbolic_functions.cpp:
```



Namespaces

- [amici](#)

The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Functions

- int [isnan](#) (double what)
- int [isinf](#) (double what)
- double [getnan](#) ()
- double [log](#) (double x)
- double [dirac](#) (double x)
- double [heaviside](#) (double x)
- double [sign](#) (double x)
- double [max](#) (double a, double b, double c)
- double [min](#) (double a, double b, double c)
- double [dmax](#) (int id, double a, double b, double c)
- double [dmin](#) (int id, double a, double b, double c)
- double [pos_pow](#) (double base, double exponent)
- double [spline](#) (double t, int num,...)
- double [spline_pos](#) (double t, int num,...)
- double [Dspline](#) (int id, double t, int num,...)
- double [Dspline_pos](#) (int id, double t, int num,...)
- double [DDspline](#) (int id1, int id2, double t, int num,...)
- double [DDspline_pos](#) (int id1, int id2, double t, int num,...)

11.20.1 Detailed Description

This file contains definitions of various symbolic functions which

Index

__init__
 amici::ode_export::Constant, 362
 amici::ode_export::Expression, 363
 amici::ode_export::LogLikelihood, 364
 amici::ode_export::ModelQuantity, 365
 amici::ode_export::ODEExporter, 370
 amici::ode_export::ODEModel, 374
 amici::ode_export::Observable, 368
 amici::ode_export::Parameter, 387
 amici::ode_export::SigmaY, 388
 amici::ode_export::State, 389
 amici::sbml_import::SbmlImporter, 411

__repr__
 amici::ode_export::ModelQuantity, 367

~Model
 amici::Model, 160

add_component
 amici::ode_export::ODEModel, 376

adjInit
 amici::Solver, 473

adjoint
 amimodel, 529

allocateSolver
 amici::Solver, 467

allocateSolverB
 amici::Solver, 473

am_and
 am_and.m, 552

am_and.m, 551
 am_and, 552

am_eq
 am_eq.m, 553

am_eq.m, 553
 am_eq, 553

am_ge
 am_ge.m, 554

am_ge.m, 554
 am_ge, 554

am_gt
 am_gt.m, 555

am_gt.m, 554
 am_gt, 555

am_if
 am_if.m, 555

am_if.m, 555
 am_if, 555

am_le
 am_le.m, 556

am_le.m, 556
 am_le, 556

am_lt
 am_lt.m, 557

am_lt.m, 557
 am_lt, 557

am_max
 am_max.m, 558

am_max.m, 558
 am_max, 558

am_min
 am_min.m, 559

am_min.m, 559
 am_min, 559

am_or
 am_or.m, 560

am_or.m, 560
 am_or, 560

am_piecewise
 am_piecewise.m, 561

am_piecewise.m, 561
 am_piecewise, 561

am_stepfun
 am_stepfun.m, 562

am_stepfun.m, 562
 am_stepfun, 562

am_xor
 am_xor.m, 563

am_xor.m, 563
 am_xor, 563

AmiException, 85
 amici::AmiException, 85, 86

AmiVector, 88
 amici::AmiVector, 88, 89

AmiVectorArray, 98
 amici::AmiVectorArray, 99

amici, 14
 amici_blasCBlasTransToBlasTrans, 70
 amici_daxpy, 28
 amici_dgemm, 26
 amici_dgemv, 25
 amici_path, 74
 BLASLayout, 19
 BLASTranspose, 20
 checkFieldNames, 44
 checkFinite, 32
 DDspline, 65
 DDspline_pos, 65
 dbl2int, 69
 deserializeFromChar, 46
 deserializeFromString, 47
 dirac, 53
 Dmax, 56
 Dmin, 55
 Dspline, 63
 Dspline_pos, 64
 errMsgIdAndTxt, 74
 ExpData, 67
 expDataFromMatlabCall, 31
 getNaN, 59
 getReturnDataMatlabFromAmiciCall, 36

getUnscaledParameter, 35
 getValueById, 71
 heaviside, 53
 initAndAttachArray, 43
 initMatlabDiagnosisFields, 37
 initMatlabReturnFields, 36
 InternalSensitivityMethod, 21
 InterpolationType, 21
 isInf, 58
 isNaN, 57
 LinearMultistepMethod, 21
 LinearSolver, 21
 log, 52
 max, 55
 min, 54
 msgIdAndTxtFp, 19
 mxArrayToVector, 70
 NewtonStatus, 22
 NonlinearSolverIteration, 21
 operator==, 35, 48
 ParameterScaling, 20
 pi, 74
 pos_pow, 57
 printErrMsgIdAndTxt, 22
 printWarnMsgIdAndTxt, 23
 realtype, 19
 reorder, 45
 runAmiciSimulation, 23, 66
 runAmiciSimulations, 24, 68
 SecondOrderMode, 20
 SensitivityMethod, 20
 SensitivityOrder, 20
 serializeToChar, 46
 serializeToStdVec, 47
 serializeToString, 47
 setModelData, 28
 setSolverOptions, 29
 setValueById, 72
 setValueByIdRegex, 73
 setupReturnData, 30
 seval, 50
 sign, 60
 sinteg, 51
 spline, 48, 60
 spline_pos, 62
 StateOrdering, 22
 SteadyStateSensitivityMode, 22
 unscaleParameters, 33, 34
 warnMsgIdAndTxt, 74
 writeMatlabField0, 38
 writeMatlabField1, 39
 writeMatlabField2, 40
 writeMatlabField3, 41
 writeMatlabField4, 42
 amici.cpp, 564
 amici.ode_export, 75
 amici.plotting, 80
 amici.sbml_import, 81

amici::AmiException
 AmiException, 85, 86
 getBacktrace, 87
 storeBacktrace, 87
 what, 86
 amici::AmiVector
 AmiVector, 88, 89
 at, 97
 data, 90, 91
 getLength, 94
 getNVector, 92
 getVector, 93
 minus, 96
 operator=, 89
 operator[], 97
 reset, 95
 set, 96
 amici::AmiVectorArray
 AmiVectorArray, 99
 at, 101
 data, 100, 101
 getLength, 103
 getNVector, 102
 getNVectorArray, 102
 operator[], 103
 reset, 104
 amici::BackwardProblem
 BackwardProblem, 105
 getdJydx, 109
 getdxBptr, 109
 gett, 106
 getwhich, 107
 getwhichptr, 107
 getxBptr, 108
 getxQBptr, 108
 workBackwardProblem, 106
 amici::CvodeException
 CvodeException, 111
 amici::ExpData
 checkDataDimension, 134
 checkEventsDimension, 135
 checkSigmaPositivity, 135, 136
 ExpData, 113–115
 fixedParameters, 136
 fixedParametersPreequilibration, 136
 fixedParametersPresimulation, 137
 getObservedData, 122
 getObservedDataPtr, 122
 getObservedDataStdDev, 126
 getObservedDataStdDevPtr, 126
 getObservedEvents, 129
 getObservedEventsPtr, 129
 getObservedEventsStdDev, 133
 getObservedEventsStdDevPtr, 133
 getTimepoint, 119
 getTimepoints, 119
 isSetObservedData, 121
 isSetObservedDataStdDev, 125

isSetObservedEvents, 128
isSetObservedEventsStdDev, 132
nmaxevent, 117
nmaxevent_, 137
nt, 117
nytrue, 116
nytrue_, 137
nztrue, 117
nztrue_, 137
observedData, 138
observedDataStdDev, 138
observedEvents, 138
observedEventsStdDev, 138
setObservedData, 120
setObservedDataStdDev, 123–125
setObservedEvents, 127, 128
setObservedEventsStdDev, 130–132
setTimepoints, 118
ts, 137
amici::ForwardProblem
 edata, 146
 ForwardProblem, 139
 getDJydx, 143
 getDJzdx, 143
 getDiscontinuities, 142
 getNumberOfRoots, 142
 getRHSAtDiscontinuities, 141
 getRHSBeforeDiscontinuities, 142
 getRootCounter, 143
 getRootIndexes, 142
 getStateDerivativePointer, 144
 getStateDerivativeSensitivityPointer, 145
 getStatePointer, 144
 getStateSensitivity, 141
 getStateSensitivityPointer, 144
 getStatesAtDiscontinuities, 141
 getTime, 140
 model, 145
 rdata, 145
 solver, 145
 workForwardProblem, 140
amici::IDAException
 IDAException, 147
amici::IntegrationFailure
 error_code, 148
 IntegrationFailure, 148
 time, 148
amici::IntegrationFailureB
 error_code, 150
 IntegrationFailureB, 149
 time, 150
amici::Model
 ~Model, 160
 anyStateNonNegative, 279
 boost::serialization::serialize, 267
 checkFinite, 222
 clone, 160
 computeX_pos, 265
 dJrzdsigma, 275
 dJrzdz, 275
 dJydp, 272
 dJydsigma, 274
 dJydy, 274
 dJzdp, 272
 dJzdsigma, 275
 dJzdz, 275
 deltaqB, 273
 deltasx, 273
 deltax, 273
 deltaxB, 273
 drzdp, 276
 drzdx, 276
 dsigmaydp, 272
 dsigmazdp, 272
 dwdp, 277
 dwdx, 277
 dxdotdp, 273
 dydp, 276
 dydx, 276
 dzdp, 276
 dzdx, 275
 fFIM, 221
 fJDiag, 164
 fJSparse, 163
 fJrz, 187, 252
 fJv, 165
 fJy, 185, 251
 fJz, 186, 252
 fchi2, 220
 fdJrzdsigma, 191, 255
 fdJrzdz, 190, 255
 fdJydp, 193
 fdJydsigma, 188, 253
 fdJydx, 194
 fdJydy, 187, 253
 fdJzdp, 196
 fdJzdsigma, 189, 254
 fdJzdx, 196
 fdJzdz, 189, 254
 fdeltaqB, 182, 249
 fdeltasx, 180, 248
 fdeltax, 179, 246
 fdeltaxB, 181, 248
 fdrzdp, 178, 245
 fdrzdx, 179, 246
 fdsigmaydp, 183, 250
 fdsigmazdp, 184, 251
 fdwdp, 216, 256
 fdwdx, 217, 257
 fdx0, 167
 fdxdotdp, 164
 fdydp, 171, 241
 fdydx, 171, 242
 fdzdp, 177, 244
 fdzdx, 178, 245
 fixedParameters, 278

fJ, 162
fres, 219
froot, 161
frz, 175, 243
fsJy, 193
fsJz, 195
fsdx0, 169
fsigmay, 183, 250
fsigmaz, 184, 251
fsres, 220
fsrz, 176, 244
fstau, 169, 240
fsx0, 168, 240
fsx0_fixedParameters, 168, 239
fsxdot, 162
fsy, 191
fsz, 174, 243
fsz_tf, 192
fw, 215, 256
fx0, 166, 238
fx0_fixedParameters, 166, 239
fxdot, 161
fy, 170, 241
fz, 173, 242
getFixedParameterById, 205
getFixedParameterByName, 206
getFixedParameterIds, 235
getFixedParameterNames, 227
getFixedParameters, 205
getInitialStateSensitivities, 213
getInitialStates, 212
getObservableIds, 236
getObservableNames, 228
getParameterById, 230
getParameterByName, 230
getParameterIds, 229
getParameterList, 211
getParameterNames, 224
getParameterScale, 203
getParameters, 204
getReinitializeFixedParameterInitialStates, 238
getSolver, 160
getStateIds, 234
getStateIsNonNegative, 209
getStateNames, 225
getSteadyStateSensitivityMode, 237
getTimepoints, 209
getUnscaledParameters, 204
getmy, 257
getmz, 258
getrz, 262
getsrz, 264
getsx, 261
getsz, 263
gett, 222
getx, 260
gety, 259
getz, 262
h, 278
hasFixedParameterIds, 234
hasFixedParameterNames, 226
hasObservableIds, 236
hasObservableNames, 227
hasParameterIds, 228
hasParameterNames, 223
hasStateIds, 234
hasStateNames, 225
idlist, 271
initHeaviside, 198
initialize, 197
initializeStates, 197
isFixedParameterStateReinitializationAllowed, 264
J, 274
k, 202
lbw, 271
M, 277
Model, 158, 159
my, 274
mz, 274
nMaxEvent, 202
ndwdp, 270
ndwdx, 270
ne, 269
nJ, 270
nk, 201
nmaxevent, 280
nnz, 270
np, 200
nplist, 199
nt, 202
nw, 269
nx, 268
nxtrue, 268
ny, 268
nytrue, 269
nz, 269
nztrue, 269
o2mode, 271
operator=, 160
operator==, 268
originalParameters, 278
plist, 214
plist_, 278
pscale, 280
reinitializeFixedParameterInitialStates, 281
setFixedParameterById, 207
setFixedParameterByName, 208
setFixedParameters, 205
setFixedParametersByIdRegex, 207
setFixedParametersByNameRegex, 208
setInitialStateSensitivities, 213
setInitialStates, 212
setNMaxEvent, 202
setParameterById, 231
setParameterByName, 232
setParameterList, 212

setParameterScale, 203
setParameter, 204
setParametersByIdRegex, 232
setParametersByNameRegex, 233
setReinitializeFixedParameterInitialStates, 237
setStatelsNonNegative, 210
setSteadyStateSensitivityMode, 237
setT0, 214
setTimepoints, 209
sigmay, 271
sigmaz, 272
statelsNonNegative, 279
stau, 277
steadyStateSensitivityMode, 280
sx0data, 279
t, 210
t0, 213
ts, 279
tstart, 280
ubw, 270
unscaledParameters, 278
updateHeaviside, 221
updateHeavisideB, 222
w, 277
x0data, 279
x_pos_tmp, 280
z2event, 271
amici::Model_DAE
fJDiag, 290, 307
fJSparse, 287, 288, 306
fJSparseB, 289, 307
fJB, 286, 306
fJv, 291, 292, 308
fJvB, 293, 309
fdxdotdp, 300, 301, 312
fJ, 284, 285, 305
fM, 304, 313
fqBdot, 299, 311
froot, 294, 295, 309
fsxdot, 302, 303, 312
fxBdot, 298, 310
fxdot, 295, 296, 310
getSolver, 305
Model_DAE, 283
amici::Model_ODE
fJDiag, 322, 323, 338
fJSparse, 319, 320, 337
fJSparseB, 321, 338
fJB, 318, 337
fJv, 324, 325, 339
fJvB, 326, 340
fdxdotdp, 332, 333, 342
fJ, 317, 318, 336
fqBdot, 332, 342
froot, 327, 328, 340
fsxdot, 334, 335, 343
fxBdot, 331, 341
fxdot, 328, 329, 341
getSolver, 336
Model_ODE, 315, 316
amici::NewtonFailure
error_code, 345
NewtonFailure, 344
amici::NewtonSolver
atol, 350
computeNewtonSensis, 348
dx, 352
getSolver, 346
getStep, 347
maxlinsteps, 350
maxsteps, 350
model, 351
NewtonSolver, 346
prepareLinearSystem, 349
rdata, 351
rtol, 351
solveLinearSystem, 349
t, 351
x, 352
xdot, 351
amici::NewtonSolverDense
NewtonSolverDense, 353
prepareLinearSystem, 354
solveLinearSystem, 353
amici::NewtonSolverIterative
linsolveSPBCG, 357
NewtonSolverIterative, 355
prepareLinearSystem, 357
solveLinearSystem, 356
amici::NewtonSolverSparse
NewtonSolverSparse, 359
prepareLinearSystem, 360
solveLinearSystem, 360
amici::ReturnData
applyChainRuleFactorToSimulationResults, 395
boost::serialization::serialize, 396
chi2, 404
FIM, 400
invalidate, 394
invalidateLLH, 395
J, 397
llh, 404
ne, 406
newton_cpu_time, 402
newton_maxsteps, 407
newton_numlinsteps, 403
newton_numsteps, 402
newton_status, 402
nJ, 407
nk, 405
nmaxevent, 407
np, 405
nplist, 407
nt, 407
numerrtestfails, 401
numerrtestfailsB, 401

numnonlinsolvconvfails, 401
 numnonlinsolvconvfailsB, 402
 numrhsevals, 401
 numrhsevalsB, 401
 numsteps, 400
 numstepsB, 400
 nx, 405
 nxtrue, 405
 ny, 406
 nytrue, 406
 nz, 406
 nztrue, 406
 o2mode, 408
 order, 402
 pscale, 408
 res, 400
 ReturnData, 392, 394
 rz, 398
 s2lh, 404
 s2rz, 398
 sensi, 408
 sensi_meth, 408
 sigmay, 399
 sigmaz, 397
 slh, 404
 sres, 400
 srz, 398
 ssigmay, 399
 ssigmaz, 398
 status, 405
 sx, 399
 sx0, 404
 sy, 399
 sz, 397
 t_steadystate, 403
 ts, 397
 wrms_sensi_steadystate, 403
 wrms_steadystate, 403
 x, 398
 x0, 403
 xdot, 397
 y, 399
 z, 397
 amici::SetupFailure
 SetupFailure, 424
 amici::Solver
 adjInit, 473
 allocateSolver, 467
 allocateSolverB, 473
 applyQuadTolerancesASA, 490
 applySensitivityTolerances, 490
 applyTolerances, 488
 applyTolerancesASA, 489
 applyTolerancesFSA, 488
 band, 476
 bandB, 476
 binit, 463
 boost::serialization::serialize, 491
 calcICB, 435
 calcIC, 435
 clone, 429
 dense, 475
 denseB, 475
 diag, 477
 diagB, 477
 getAbsoluteTolerance, 445
 getAbsoluteToleranceQuadratures, 449
 getAbsoluteToleranceSensi, 447
 getAbsoluteToleranceSteadyState, 451
 getAbsoluteToleranceSteadyStateSensi, 453
 getAdjBmem, 487
 getAdjMallocDone, 487
 getDiagnosis, 432
 getDiagnosisB, 433
 getDky, 473
 getInternalSensitivityMethod, 461
 getInterpolationType, 458
 getLastOrder, 482
 getLinearMultistepMethod, 456
 getLinearSolver, 460
 getMallocDone, 486
 getMaxSteps, 454
 getMaxStepsBackwardProblem, 455
 getModel, 486
 getNewtonMaxLinearSteps, 442
 getNewtonMaxSteps, 440
 getNewtonPreequilibration, 441
 getNonlinearSolverIteration, 457
 getNumErrTestFails, 481
 getNumNonlinSolvConvFails, 482
 getNumRhsEvals, 481
 getNumSteps, 480
 getQuadB, 438
 getRelativeTolerance, 444
 getRelativeToleranceQuadratures, 448
 getRelativeToleranceSensi, 447
 getRelativeToleranceSteadyState, 450
 getRelativeToleranceSteadyStateSensi, 452
 getRootInfo, 434
 getSens, 432
 getSensitivityMethod, 439
 getSensitivityOrder, 443
 getStabilityLimitFlag, 459
 getStateOrdering, 458
 getB, 437
 init, 462
 initializeLinearSolver, 483
 initializeLinearSolverB, 484
 interpType, 493
 ism, 492
 iter, 493
 klu, 479
 kluSetOrdering, 479
 kluSetOrderingB, 480
 klub, 480
 lmm, 492

maxsteps, 493
nplist, 485
nx, 485
operator==, 491
qbinit, 463
quadRelInitB, 438
quadSStolerancesB, 474
relInit, 434
relInitB, 437
rootInit, 464
sensInit1, 464
sensRelInit, 434
setAbsoluteTolerance, 446
setAbsoluteToleranceQuadratures, 449
setAbsoluteToleranceSensi, 448
setAbsoluteToleranceSteadyState, 452
setAbsoluteToleranceSteadyStateSensi, 454
setBandJacFn, 465
setBandJacFnB, 466
setDenseJacFn, 465
setDenseJacFnB, 465
setErrHandlerFn, 469
setId, 472
setInternalSensitivityMethod, 462
setInterpolationType, 458
setJacTimesVecFn, 465
setJacTimesVecFnB, 466
setLinearMultistepMethod, 456
setLinearSolver, 461
setMaxNumSteps, 470
setMaxNumStepsB, 471
setMaxSteps, 454
setMaxStepsBackwardProblem, 455
setNewtonMaxLinearSteps, 442
setNewtonMaxSteps, 440
setNewtonPreequilibration, 441
setNonlinearSolverIteration, 457
setQuadErrConB, 469
setRelativeTolerance, 444
setRelativeToleranceQuadratures, 449
setRelativeToleranceSensi, 447
setRelativeToleranceSteadyState, 451
setRelativeToleranceSteadyStateSensi, 453
setSStolerances, 467
setSStolerancesB, 474
setSensErrCon, 468
setSensParams, 473
setSensSStolerances, 468
setSensitivityMethod, 439
setSensitivityOrder, 443
setSparseJacFn, 465
setSparseJacFnB, 466
setStabLimDet, 471
setStabLimDetB, 472
setStabilityLimitFlag, 460
setStateOrdering, 459
setStopTime, 437
setSuppressAlg, 473
setUserData, 469
setUserDataB, 470
setup, 429
setupAMIB, 430
solve, 435
solveB, 437
solveF, 436
Solver, 428
solverMemory, 492
solverMemoryB, 492
solverWasCalled, 492
spbcg, 478
spbcgB, 478
spgmr, 477
spgmrB, 477
sptfqmr, 478
sptfqmrB, 479
turnOffRootFinding, 438
wrapErrHandlerFn, 467
amici::SteadystateProblem
 applyNewtonsMethod, 497
 checkConvergence, 496
 createSteadystateSimSolver, 499
 getSteadystateSimulation, 498
 getWrmsNorm, 495
 SteadystateProblem, 494
 workSteadyStateProblem, 494
 writeNewtonOutput, 498
amici::ode_export
 applyTemplate, 77
 functions, 78
 getSymbolicDiagonal, 77
 multiobs_functions, 79
 sanitize_basic_sympy, 78
 sensi_functions, 79
 sparse_functions, 79
 symbol_to_type, 79
 var_in_function_signature, 76
amici::ode_export::Constant
 __init__, 362
amici::ode_export::Expression
 __init__, 363
amici::ode_export::LogLikelihood
 __init__, 364
amici::ode_export::ModelQuantity
 __init__, 365
 __repr__, 367
amici::ode_export::ODEExporter
 __init__, 370
 compileModel, 371
 generateModelCode, 370
 setName, 373
 setPaths, 372
amici::ode_export::ODEModel
 __init__, 374
 add_component, 376
 colptr, 382
 eq, 380

generateBasicVariables, 384
 import_from_sbml_importer, 375
 name, 384
 nk, 378
 np, 378
 nx, 376
 ny, 377
 rowval, 382
 sparseeq, 381
 sparsesym, 380
 sym, 379
 symNames, 385
 val, 383
 amici::ode_export::Observable
 __init__, 368
 amici::ode_export::Parameter
 __init__, 387
 amici::ode_export::SigmaY
 __init__, 388
 amici::ode_export::State
 __init__, 389
 amici::plotting
 plotObservableTrajectories, 80
 plotStateTrajectories, 80
 amici::sbml_import
 assignmentRules2observables, 82
 constantSpeciesToParameters, 83
 default_symbols, 84
 getRuleVars, 82
 l2s, 84
 replaceLogAB, 83
 amici::sbml_import::SbmlImporter
 __init__, 411
 checkLibSBMLErrors, 412
 checkSupport, 415
 cleanReservedSymbols, 422
 loadSBMLFile, 412
 processCompartments, 417
 processObservables, 419
 processParameters, 416
 processReactions, 417
 processRules, 418
 processSBML, 414
 processSpecies, 416
 processTime, 419
 processVolumeConversion, 418
 replaceInAllExpressions, 421
 replaceSpecialConstants, 422
 reset_symbols, 411
 sbml2amici, 413
 amici_blasCBlasTransToBlasTrans
 amici, 70
 amici_daxpy
 amici, 28
 amici_dgemm
 amici, 26
 amici_dgemv
 amici, 25
 amici_path
 amici, 74
 amidata, 500
 amidata, 501
 condition, 504
 conditionPreequilibration, 504
 ne, 502
 nk, 503
 nt, 502
 ny, 502
 nz, 502
 Sigma_Y, 503
 Sigma_Z, 504
 t, 503
 Y, 503
 Z, 504
 amievent, 505
 amievent, 505
 bolus, 507
 hflag, 507
 setHflag, 506
 trigger, 506
 z, 507
 amifun, 508
 amifun, 509
 argstr, 514
 cvar, 514
 deps, 514
 funstr, 514
 gccode, 510
 getArgs, 511
 getCVar, 512
 getDeps, 511
 getNVecs, 512
 getSensiFlag, 512
 getSyms, 512
 nvecs, 514
 sensiflag, 515
 strsym, 513
 strsym_old, 513
 sym, 513
 sym_noopt, 513
 writeCcode, 510
 writeCcode_sensi, 509
 writeMcode, 510
 amimodel, 515
 adjoint, 529
 amimodel, 518
 augmento2, 525
 augmento2vec, 525
 cfun, 537
 checkDeps, 524
 colptrs, 534
 colptrsB, 535
 compileAndLinkModel, 526
 compileC, 521
 coptim, 536
 debug, 529

event, 528
forward, 529
fun, 528
fun, 535
generateMatlabWrapper, 527
generateC, 520
generateM, 521
getFun, 522
HTable, 529
id, 533
lbw, 533
loadOldHashes, 525
makeEvents, 523
makeSyms, 523
maxflag, 538
mfun, 535
minflag, 538
modelname, 528
ndwdp, 538
ndwdx, 538
nevent, 532
ng, 532
nk, 531
nnz, 533
np, 531
nw, 538
nx, 530
nxtrue, 530
ny, 531
nytrue, 531
nz, 532
nztrue, 532
o2flag, 537
param, 536
parseModel, 520
recompile, 536
rowvals, 534
rowvalsB, 535
sparseidx, 534
sparseidxB, 534
splineflag, 537
sym, 528
t0, 530
ubw, 533
update modelName, 519
updateRHS, 519
updateWrapPath, 520
wrap_path, 536
wtype, 530
z2event, 537
amioption, 539
amioption, 541
atol, 541
interpType, 544
ism, 544
iter, 543
linsol, 543
Imm, 543
maxsteps, 541
maxstepsB, 542
newton_maxlinsteps, 546
newton_maxsteps, 546
newton_preq, 546
nmaxevent, 545
ordering, 545
pscale, 547
quad_atol, 542
quad_rtol, 542
rtol, 541
sens_ind, 543
sensi, 544
sensi_meth, 544
ss, 545
ss_atol, 542
ss_rtol, 542
stldet, 544
sx0, 545
tstart, 543
x0, 545
z2event, 546
amised, 547
amised, 548
datasym, 550
model, 548
modelname, 549
outputcount, 549
sedml, 549
varidx, 549
varsym, 550
amiwrap
 amiwrap.m, 564
amiwrap.m, 564
 amiwrap, 564
anyStateNonNegative
 amici::Model, 279
applyChainRuleFactorToSimulationResults
 amici::ReturnData, 395
applyNewtonsMethod
 amici::SteadystateProblem, 497
applyQuadTolerancesASA
 amici::Solver, 490
applySensitivityTolerances
 amici::Solver, 490
applyTemplate
 amici::ode_export, 77
applyTolerances
 amici::Solver, 488
applyTolerancesASA
 amici::Solver, 489
applyTolerancesFSA
 amici::Solver, 488
argstr
 amifun, 514
assignmentRules2observables
 amici::sbml_import, 82
at

amici::AmiVector, 97
 amici::AmiVectorArray, 101
 atol
 amici::NewtonSolver, 350
 amioption, 541
 augmento2
 amimodel, 525
 augmento2vec
 amimodel, 525

 BLASLayout
 amici, 19
 BLASTranspose
 amici, 20
 BackwardProblem, 104
 amici::BackwardProblem, 105
 band
 amici::Solver, 476
 bandB
 amici::Solver, 476
 binit
 amici::Solver, 463
 bolus
 amievent, 507
 boost::serialization::serialize
 amici::Model, 267
 amici::ReturnData, 396
 amici::Solver, 491

 calcICB
 amici::Solver, 435
 calcIC
 amici::Solver, 435
 cblas.cpp, 565
 cfun
 amimodel, 537
 checkConvergence
 amici::SteadystateProblem, 496
 checkDataDimension
 amici::ExpData, 134
 checkDeps
 amimodel, 524
 checkEventsDimension
 amici::ExpData, 135
 checkFieldNames
 amici, 44
 checkFinite
 amici, 32
 amici::Model, 222
 checkLibSBMLErrors
 amici::sbml_import::SbmlImporter, 412
 checkSigmaPositivity
 amici::ExpData, 135, 136
 checkSupport
 amici::sbml_import::SbmlImporter, 415
 chi2
 amici::ReturnData, 404
 cleanReservedSymbols
 amici::sbml_import::SbmlImporter, 422

 clone
 amici::Model, 160
 amici::Solver, 429
 colptr
 amici::ode_export::ODEModel, 382
 colptrs
 amimodel, 534
 colptrsB
 amimodel, 535
 compileAndLinkModel
 amimodel, 526
 compileModel
 amici::ode_export::ODEExporter, 371
 compileC
 amimodel, 521
 computeNewtonSensis
 amici::NewtonSolver, 348
 computeX_pos
 amici::Model, 265
 condition
 amidata, 504
 conditionPreequilibration
 amidata, 504
 Constant, 361
 constantSpeciesToParameters
 amici::sbml_import, 83
 coptim
 amimodel, 536
 createSteadystateSimSolver
 amici::SteadystateProblem, 499
 cvar
 amifun, 514
 CvodeException, 110
 amici::CvodeException, 111

 DDspline
 amici, 65
 DDspline_pos
 amici, 65
 dJrzdsigma
 amici::Model, 275
 dJrzdz
 amici::Model, 275
 dJydp
 amici::Model, 272
 dJydsigma
 amici::Model, 274
 dJydy
 amici::Model, 274
 dJzdp
 amici::Model, 272
 dJzdsigma
 amici::Model, 275
 dJzdz
 amici::Model, 275
 data
 amici::AmiVector, 90, 91
 amici::AmiVectorArray, 100, 101
 datasym

amised, 550
dbl2int
 amici, 69
debug
 amimodel, 529
default_symbols
 amici::sbml_import, 84
deltaqB
 amici::Model, 273
deltasx
 amici::Model, 273
deltax
 amici::Model, 273
deltaxB
 amici::Model, 273
dense
 amici::Solver, 475
denseB
 amici::Solver, 475
deps
 amifun, 514
deserializeFromChar
 amici, 46
deserializeFromString
 amici, 47
diag
 amici::Solver, 477
diagB
 amici::Solver, 477
dirac
 amici, 53
Dmax
 amici, 56
Dmin
 amici, 55
drzdp
 amici::Model, 276
drzdx
 amici::Model, 276
dsigmaydp
 amici::Model, 272
dsigmazdp
 amici::Model, 272
Dspline
 amici, 63
Dspline_pos
 amici, 64
dwdp
 amici::Model, 277
dwdx
 amici::Model, 277
dx
 amici::NewtonSolver, 352
dxdotdp
 amici::Model, 273
dydp
 amici::Model, 276
dydx
 amici::Model, 276
amici::Model, 276
dzdp
 amici::Model, 276
dzdx
 amici::Model, 275
edata
 amici::ForwardProblem, 146
eq
 amici::ode_export::ODEModel, 380
errMsgIdAndTxt
 amici, 74
error_code
 amici::IntegrationFailure, 148
 amici::IntegrationFailureB, 150
 amici::NewtonFailure, 345
event
 amimodel, 528
ExpData, 111
 amici, 67
 amici::ExpData, 113–115
expDataFromMatlabCall
 amici, 31
Expression, 362
fFIM
 amici::Model, 221
FIM
 amici::ReturnData, 400
fJDiag
 amici::Model, 164
 amici::Model_DAE, 290, 307
 amici::Model_ODE, 322, 323, 338
fJSparse
 amici::Model, 163
 amici::Model_DAE, 287, 288, 306
 amici::Model_ODE, 319, 320, 337
fJSparseB
 amici::Model_DAE, 289, 307
 amici::Model_ODE, 321, 338
fJB
 amici::Model_DAE, 286, 306
 amici::Model_ODE, 318, 337
fJrz
 amici::Model, 187, 252
fJv
 amici::Model, 165
 amici::Model_DAE, 291, 292, 308
 amici::Model_ODE, 324, 325, 339
fJvB
 amici::Model_DAE, 293, 309
 amici::Model_ODE, 326, 340
fJy
 amici::Model, 185, 251
fJz
 amici::Model, 186, 252
fchi2
 amici::Model, 220
fdJrzdsigma

amici::Model, 191, 255
fdJrzdz
 amici::Model, 190, 255
fdJydp
 amici::Model, 193
fdJydsigma
 amici::Model, 188, 253
fdJydx
 amici::Model, 194
fdJydy
 amici::Model, 187, 253
fdJzdp
 amici::Model, 196
fdJzdsigma
 amici::Model, 189, 254
fdJzdx
 amici::Model, 196
fdJzdz
 amici::Model, 189, 254
fdeltaqB
 amici::Model, 182, 249
fdeltaxs
 amici::Model, 180, 248
fdeltax
 amici::Model, 179, 246
fdeltaxB
 amici::Model, 181, 248
fdrzdp
 amici::Model, 178, 245
fdrzdx
 amici::Model, 179, 246
fdsigmaydp
 amici::Model, 183, 250
fdsigmazdp
 amici::Model, 184, 251
fdwdp
 amici::Model, 216, 256
fdwdx
 amici::Model, 217, 257
fdx0
 amici::Model, 167
fdxdotdp
 amici::Model, 164
 amici::Model_DAE, 300, 301, 312
 amici::Model_ODE, 332, 333, 342
fdydp
 amici::Model, 171, 241
fdydx
 amici::Model, 171, 242
fdzdp
 amici::Model, 177, 244
fdzdx
 amici::Model, 178, 245
fixedParameters
 amici::ExpData, 136
 amici::Model, 278
fixedParametersPreequilibration
 amici::ExpData, 136
fixedParametersPresimulation
 amici::ExpData, 137
fJ
 amici::Model, 162
 amici::Model_DAE, 284, 285, 305
 amici::Model_ODE, 317, 318, 336
fM
 amici::Model_DAE, 304, 313
forward
 amimodel, 529
ForwardProblem, 138
 amici::ForwardProblem, 139
fqBdot
 amici::Model_DAE, 299, 311
 amici::Model_ODE, 332, 342
fres
 amici::Model, 219
froot
 amici::Model, 161
 amici::Model_DAE, 294, 295, 309
 amici::Model_ODE, 327, 328, 340
frz
 amici::Model, 175, 243
fsJy
 amici::Model, 193
fsJz
 amici::Model, 195
fsdx0
 amici::Model, 169
fsigmay
 amici::Model, 183, 250
fsigmaz
 amici::Model, 184, 251
fsres
 amici::Model, 220
fsrz
 amici::Model, 176, 244
fstau
 amici::Model, 169, 240
fsx0
 amici::Model, 168, 240
fsx0_fixedParameters
 amici::Model, 168, 239
fsxdot
 amici::Model, 162
 amici::Model_DAE, 302, 303, 312
 amici::Model_ODE, 334, 335, 343
fsy
 amici::Model, 191
fsz
 amici::Model, 174, 243
fsz_tf
 amici::Model, 192
fun
 amimodel, 528
functions
 amici::ode_export, 78
fun

amimodel, 535
funstr
 amifun, 514
fw
 amici::Model, 215, 256
fx0
 amici::Model, 166, 238
fx0_fixedParameters
 amici::Model, 166, 239
fxBdot
 amici::Model_DAE, 298, 310
 amici::Model_ODE, 331, 341
fxdot
 amici::Model, 161
 amici::Model_DAE, 295, 296, 310
 amici::Model_ODE, 328, 329, 341
fy
 amici::Model, 170, 241
fz
 amici::Model, 173, 242

gccode
 amifun, 510
generateBasicVariables
 amici::ode_export::ODEModel, 384
generateMatlabWrapper
 amimodel, 527
generateModelCode
 amici::ode_export::ODEExporter, 370
generateC
 amimodel, 520
generateM
 amimodel, 521
getAbsoluteTolerance
 amici::Solver, 445
getAbsoluteToleranceQuadratures
 amici::Solver, 449
getAbsoluteToleranceSensi
 amici::Solver, 447
getAbsoluteToleranceSteadyState
 amici::Solver, 451
getAbsoluteToleranceSteadyStateSensi
 amici::Solver, 453
getAdjBmem
 amici::Solver, 487
getAdjMallocDone
 amici::Solver, 487
getArgs
 amifun, 511
getBacktrace
 amici::AmiException, 87
getCVar
 amifun, 512
getDJydx
 amici::ForwardProblem, 143
getDJzdx
 amici::ForwardProblem, 143
getDepS
 amifun, 511
getDiagnosis
 amici::Solver, 432
getDiagnosisB
 amici::Solver, 433
getDiscontinuities
 amici::ForwardProblem, 142
getDky
 amici::Solver, 473
getFixedParameterByld
 amici::Model, 205
getFixedParameterByName
 amici::Model, 206
getFixedParameterIds
 amici::Model, 235
getFixedParameterNames
 amici::Model, 227
getFixedParameters
 amici::Model, 205
getFun
 amimodel, 522
getInitialStateSensitivities
 amici::Model, 213
getInitialStates
 amici::Model, 212
getInternalSensitivityMethod
 amici::Solver, 461
getInterpolationType
 amici::Solver, 458
getLastOrder
 amici::Solver, 482
getLength
 amici::AmiVector, 94
 amici::AmiVectorArray, 103
getLinearMultistepMethod
 amici::Solver, 456
getLinearSolver
 amici::Solver, 460
getMallocDone
 amici::Solver, 486
getMaxSteps
 amici::Solver, 454
getMaxStepsBackwardProblem
 amici::Solver, 455
getModel
 amici::Solver, 486
getNVecs
 amifun, 512
getNVector
 amici::AmiVector, 92
 amici::AmiVectorArray, 102
getNVectorArray
 amici::AmiVectorArray, 102
getNaN
 amici, 59
getNewtonMaxLinearSteps
 amici::Solver, 442
getNewtonMaxSteps
 amici::Solver, 440

getNewtonPreequilibration
 amici::Solver, 441
 getNonlinearSolverIteration
 amici::Solver, 457
 getNumErrTestFails
 amici::Solver, 481
 getNumNonlinSolvConvFails
 amici::Solver, 482
 getNumRhsEvals
 amici::Solver, 481
 getNumSteps
 amici::Solver, 480
 getNumberOfRoots
 amici::ForwardProblem, 142
 getObservableIds
 amici::Model, 236
 getObservableNames
 amici::Model, 228
 getObservedData
 amici::ExpData, 122
 getObservedDataPtr
 amici::ExpData, 122
 getObservedDataStdDev
 amici::ExpData, 126
 getObservedDataStdDevPtr
 amici::ExpData, 126
 getObservedEvents
 amici::ExpData, 129
 getObservedEventsPtr
 amici::ExpData, 129
 getObservedEventsStdDev
 amici::ExpData, 133
 getObservedEventsStdDevPtr
 amici::ExpData, 133
 getParameterByIdx
 amici::Model, 230
 getParameterByName
 amici::Model, 230
 getParameterIds
 amici::Model, 229
 getParameterList
 amici::Model, 211
 getParameterNames
 amici::Model, 224
 getParameterScale
 amici::Model, 203
 getParameters
 amici::Model, 204
 getQuadB
 amici::Solver, 438
 getRHSAtDiscontinuities
 amici::ForwardProblem, 141
 getRHSBeforeDiscontinuities
 amici::ForwardProblem, 142
 getReinitializeFixedParameterInitialStates
 amici::Model, 238
 getRelativeTolerance
 amici::Solver, 444
 getRelativeToleranceQuadratures
 amici::Solver, 448
 getRelativeToleranceSensi
 amici::Solver, 447
 getRelativeToleranceSteadyState
 amici::Solver, 450
 getRelativeToleranceSteadyStateSensi
 amici::Solver, 452
 getReturnDataMatlabFromAmiciCall
 amici, 36
 getRootCounter
 amici::ForwardProblem, 143
 getRootIndexes
 amici::ForwardProblem, 142
 getRootInfo
 amici::Solver, 434
 getRuleVars
 amici::sbml_import, 82
 getSens
 amici::Solver, 432
 getSensiFlag
 amifun, 512
 getSensitivityMethod
 amici::Solver, 439
 getSensitivityOrder
 amici::Solver, 443
 getSolver
 amici::Model, 160
 amici::Model_DAE, 305
 amici::Model_ODE, 336
 amici::NewtonSolver, 346
 getStabilityLimitFlag
 amici::Solver, 459
 getStateDerivativePointer
 amici::ForwardProblem, 144
 getStateDerivativeSensitivityPointer
 amici::ForwardProblem, 145
 getStateIds
 amici::Model, 234
 getStateIsNonNegative
 amici::Model, 209
 getStateNames
 amici::Model, 225
 getStateOrdering
 amici::Solver, 458
 getStatePointer
 amici::ForwardProblem, 144
 getStateSensitivity
 amici::ForwardProblem, 141
 getStateSensitivityPointer
 amici::ForwardProblem, 144
 getStatesAtDiscontinuities
 amici::ForwardProblem, 141
 getSteadyStateSensitivityMode
 amici::Model, 237
 getSteadystateSimulation
 amici::SteadystateProblem, 498
 getStep

amici::NewtonSolver, 347
getSymbolicDiagonal
 amici::ode_export, 77
getSyms
 amifun, 512
getTime
 amici::ForwardProblem, 140
getTimepoint
 amici::ExpData, 119
getTimepoints
 amici::ExpData, 119
 amici::Model, 209
getUnscaledParameter
 amici, 35
getUnscaledParameters
 amici::Model, 204
getValueById
 amici, 71
getVector
 amici::AmiVector, 93
getWrmsNorm
 amici::SteadystateProblem, 495
getB
 amici::Solver, 437
getdJydx
 amici::BackwardProblem, 109
getdxBptr
 amici::BackwardProblem, 109
getmy
 amici::Model, 257
getmz
 amici::Model, 258
getoptimized
 optsym, 551
getrz
 amici::Model, 262
getsrz
 amici::Model, 264
getsx
 amici::Model, 261
getsz
 amici::Model, 263
gett
 amici::BackwardProblem, 106
 amici::Model, 222
getwhich
 amici::BackwardProblem, 107
getwhichptr
 amici::BackwardProblem, 107
getx
 amici::Model, 260
getxBptr
 amici::BackwardProblem, 108
getxQBptr
 amici::BackwardProblem, 108
gety
 amici::Model, 259
getz
 amici::Model, 262
h
 amici::Model, 278
HTable
 amimodel, 529
hasFixedParameterIds
 amici::Model, 234
hasFixedParameterNames
 amici::Model, 226
hasObservableIds
 amici::Model, 236
hasObservableNames
 amici::Model, 227
hasParameterIds
 amici::Model, 228
hasParameterNames
 amici::Model, 223
hasStatelids
 amici::Model, 234
hasStateNames
 amici::Model, 225
heaviside
 amici, 53
hflag
 amievent, 507
IDAException, 146
 amici::IDAException, 147
id
 amimodel, 533
idlist
 amici::Model, 271
import_from_sbml_importer
 amici::ode_export::ODEModel, 375
init
 amici::Solver, 462
initAndAttachArray
 amici, 43
initHeaviside
 amici::Model, 198
initMatlabDiagnosisFields
 amici, 37
initMatlabReturnFields
 amici, 36
initialize
 amici::Model, 197
initializeLinearSolver
 amici::Solver, 483
initializeLinearSolverB
 amici::Solver, 484
initializeStates
 amici::Model, 197
IntegrationFailure, 147
 amici::IntegrationFailure, 148
IntegrationFailureB, 149
 amici::IntegrationFailureB, 149
interface_matlab.cpp, 566
 mexFunction, 567

InternalSensitivityMethod
 amici, 21
 interpType
 amici::Solver, 493
 amioption, 544
 InterpolationType
 amici, 21
 invalidate
 amici::ReturnData, 394
 invalidateLLH
 amici::ReturnData, 395
 isFixedParameterStateReinitializationAllowed
 amici::Model, 264
 isInf
 amici, 58
 isNaN
 amici, 57
 isSetObservedData
 amici::ExpData, 121
 isSetObservedDataStdDev
 amici::ExpData, 125
 isSetObservedEvents
 amici::ExpData, 128
 isSetObservedEventsStdDev
 amici::ExpData, 132
 ism
 amici::Solver, 492
 amioption, 544
 iter
 amici::Solver, 493
 amioption, 543

 J
 amici::Model, 274
 amici::ReturnData, 397

 k
 amici::Model, 202
 klu
 amici::Solver, 479
 kluSetOrdering
 amici::Solver, 479
 kluSetOrderingB
 amici::Solver, 480
 kluB
 amici::Solver, 480

 l2s
 amici::sbml_import, 84
 lbw
 amici::Model, 271
 amimodel, 533
 LinearMultistepMethod
 amici, 21
 LinearSolver
 amici, 21
 linsol
 amioption, 543
 linsolveSPBCG

 amici::NewtonSolverIterative, 357
 llh
 amici::ReturnData, 404
 lmm
 amici::Solver, 492
 amioption, 543
 loadOldHashes
 amimodel, 525
 loadSBMLFile
 amici::sbml_import::SbmlImporter, 412
 log
 amici, 52
 LogLikelihood, 364

 M
 amici::Model, 277
 makeEvents
 amimodel, 523
 makeSyms
 amimodel, 523
 max
 amici, 55
 maxflag
 amimodel, 538
 maxlinsteps
 amici::NewtonSolver, 350
 maxsteps
 amici::NewtonSolver, 350
 amici::Solver, 493
 amioption, 541
 maxstepsB
 amioption, 542
 mexFunction
 interface_matlab.cpp, 567
 mfun
 amimodel, 535
 min
 amici, 54
 minflag
 amimodel, 538
 minus
 amici::AmiVector, 96
 Model, 150
 amici::Model, 158, 159
 model
 amici::ForwardProblem, 145
 amici::NewtonSolver, 351
 amised, 548
 Model_DAE, 281
 amici::Model_DAE, 283
 Model_ODE, 314
 amici::Model_ODE, 315, 316
 ModelQuantity, 365
 modelName
 amimodel, 528
 amised, 549
 msgIdAndTxtFp
 amici, 19
 multiobs_functions

amici::ode_export, 79
mxArrayToVector
 amici, 70
my
 amici::Model, 274
mz
 amici::Model, 274

nMaxEvent
 amici::Model, 202
name
 amici::ode_export::ODEModel, 384
ndwdp
 amici::Model, 270
 amimodel, 538
ndwdx
 amici::Model, 270
 amimodel, 538
ne
 amici::Model, 269
 amici::ReturnData, 406
 amidata, 502
nevent
 amimodel, 532
newton_cpu_time
 amici::ReturnData, 402
newton_maxlinsteps
 amioption, 546
newton_maxsteps
 amici::ReturnData, 407
 amioption, 546
newton_numlinsteps
 amici::ReturnData, 403
newton_numsteps
 amici::ReturnData, 402
newton_preq
 amioption, 546
newton_status
 amici::ReturnData, 402
NewtonFailure, 344
 amici::NewtonFailure, 344
NewtonSolver, 345
 amici::NewtonSolver, 346
NewtonSolverDense, 352
 amici::NewtonSolverDense, 353
NewtonSolverIterative, 355
 amici::NewtonSolverIterative, 355
NewtonSolverSparse, 359
 amici::NewtonSolverSparse, 359
NewtonStatus
 amici, 22
ng
 amimodel, 532
nJ
 amici::Model, 270
 amici::ReturnData, 407
nk
 amici::Model, 201
 amici::ReturnData, 405
 amici::ode_export::ODEModel, 378
 amidata, 503
 amimodel, 531
nmaxevent
 amici::ExpData, 117
 amici::Model, 280
 amici::ReturnData, 407
 amioption, 545
nmaxevent_
 amici::ExpData, 137
nnz
 amici::Model, 270
 amimodel, 533
NonlinearSolverIteration
 amici, 21
np
 amici::Model, 200
 amici::ReturnData, 405
 amici::ode_export::ODEModel, 378
 amimodel, 531
nplist
 amici::Model, 199
 amici::ReturnData, 407
 amici::Solver, 485
nt
 amici::ExpData, 117
 amici::Model, 202
 amici::ReturnData, 407
 amidata, 502
numerrtestfails
 amici::ReturnData, 401
numerrtestfailsB
 amici::ReturnData, 401
numnonlinconvfails
 amici::ReturnData, 401
numnonlinconvfailsB
 amici::ReturnData, 402
numrhsevals
 amici::ReturnData, 401
numrhsevalsB
 amici::ReturnData, 401
numsteps
 amici::ReturnData, 400
numstepsB
 amici::ReturnData, 400
nvecs
 amifun, 514
nw
 amici::Model, 269
 amimodel, 538
nx
 amici::Model, 268
 amici::ReturnData, 405
 amici::Solver, 485
 amici::ode_export::ODEModel, 376
 amimodel, 530
nxtrue
 amici::Model, 268

amici::ReturnData, 405
 amimodel, 530
 ny
 amici::Model, 268
 amici::ReturnData, 406
 amici::ode_export::ODEModel, 377
 amidata, 502
 amimodel, 531
 nytrue
 amici::ExpData, 116
 amici::Model, 269
 amici::ReturnData, 406
 amimodel, 531
 nytrue_
 amici::ExpData, 137
 nz
 amici::Model, 269
 amici::ReturnData, 406
 amidata, 502
 amimodel, 532
 nztrue
 amici::ExpData, 117
 amici::Model, 269
 amici::ReturnData, 406
 amimodel, 532
 nztrue_
 amici::ExpData, 137
 o2flag
 amimodel, 537
 o2mode
 amici::Model, 271
 amici::ReturnData, 408
 ODEExporter, 369
 ODEMModel, 373
 Observable, 368
 observedData
 amici::ExpData, 138
 observedDataStdDev
 amici::ExpData, 138
 observedEvents
 amici::ExpData, 138
 observedEventsStdDev
 amici::ExpData, 138
 operator=
 amici::AmiVector, 89
 amici::Model, 160
 operator==
 amici, 35, 48
 amici::Model, 268
 amici::Solver, 491
 operator[]
 amici::AmiVector, 97
 amici::AmiVectorArray, 103
 optsym, 550
 getoptimized, 551
 optsym, 551
 order
 amici::ReturnData, 402
 ordering
 amioption, 545
 originalParameters
 amici::Model, 278
 outputcount
 amised, 549
 param
 amimodel, 536
 Parameter, 386
 ParameterScaling
 amici, 20
 parseModel
 amimodel, 520
 pi
 amici, 74
 plist
 amici::Model, 214
 plist_
 amici::Model, 278
 plotObservableTrajectories
 amici::plotting, 80
 plotStateTrajectories
 amici::plotting, 80
 pos_pow
 amici, 57
 prepareLinearSystem
 amici::NewtonSolver, 349
 amici::NewtonSolverDense, 354
 amici::NewtonSolverIterative, 357
 amici::NewtonSolverSparse, 360
 printErrMsgIdAndTxt
 amici, 22
 printWarnErrMsgIdAndTxt
 amici, 23
 processCompartments
 amici::sbml_import::SbmlImporter, 417
 processObservables
 amici::sbml_import::SbmlImporter, 419
 processParameters
 amici::sbml_import::SbmlImporter, 416
 processReactions
 amici::sbml_import::SbmlImporter, 417
 processRules
 amici::sbml_import::SbmlImporter, 418
 processSBML
 amici::sbml_import::SbmlImporter, 414
 processSpecies
 amici::sbml_import::SbmlImporter, 416
 processTime
 amici::sbml_import::SbmlImporter, 419
 processVolumeConversion
 amici::sbml_import::SbmlImporter, 418
 pscale
 amici::Model, 280
 amici::ReturnData, 408
 amioption, 547
 qbinit

amici::Solver, 463
quad_atol
 amioption, 542
quad_rtol
 amioption, 542
quadReInitB
 amici::Solver, 438
quadSStolerancesB
 amici::Solver, 474

rdata
 amici::ForwardProblem, 145
 amici::NewtonSolver, 351
relinit
 amici::Solver, 434
relinitB
 amici::Solver, 437
realtype
 amici, 19
recompile
 amimodel, 536
reinitializeFixedParameterInitialStates
 amici::Model, 281
reorder
 amici, 45
replaceInAllExpressions
 amici::sbml_import::SbmlImporter, 421
replaceLogAB
 amici::sbml_import, 83
replaceSpecialConstants
 amici::sbml_import::SbmlImporter, 422
res
 amici::ReturnData, 400
reset
 amici::AmiVector, 95
 amici::AmiVectorArray, 104
reset_symbols
 amici::sbml_import::SbmlImporter, 411
ReturnData, 391
 amici::ReturnData, 392, 394
rootInit
 amici::Solver, 464
rowval
 amici::ode_export::ODEModel, 382
rowvals
 amimodel, 534
rowvalsB
 amimodel, 535
rtol
 amici::NewtonSolver, 351
 amioption, 541
runAmiciSimulation
 amici, 23, 66
runAmiciSimulations
 amici, 24, 68
rz
 amici::ReturnData, 398

s2llh
 amici::ReturnData, 404
s2rz
 amici::ReturnData, 398
SBML2AMICI.m, 568
 SBML2AMICI, 568
SBML2AMICI
 SBML2AMICI.m, 568
SBMLError, 409
sanitize_basic_sympy
 amici::ode_export, 78
sbml2amici
 amici::sbml_import::SbmlImporter, 413
SbmlImporter, 409
SecondOrderMode
 amici, 20
sedml
 amised, 549
sens_ind
 amioption, 543
sensInit1
 amici::Solver, 464
sensReInit
 amici::Solver, 434
sensi
 amici::ReturnData, 408
 amioption, 544
sensi_functions
 amici::ode_export, 79
sensi_meth
 amici::ReturnData, 408
 amioption, 544
sensiflag
 amifun, 515
SensitivityMethod
 amici, 20
SensitivityOrder
 amici, 20
serializeToChar
 amici, 46
serializeToStdVec
 amici, 47
serializeToString
 amici, 47
set
 amici::AmiVector, 96
setAbsoluteTolerance
 amici::Solver, 446
setAbsoluteToleranceQuadratures
 amici::Solver, 449
setAbsoluteToleranceSensi
 amici::Solver, 448
setAbsoluteToleranceSteadyState
 amici::Solver, 452
setAbsoluteToleranceSteadyStateSensi
 amici::Solver, 454
setBandJacFn
 amici::Solver, 465
setBandJacFnB

amici::Solver, 466
 setDenseJacFn
 amici::Solver, 465
 setDenseJacFnB
 amici::Solver, 465
 setErrHandlerFn
 amici::Solver, 469
 setFixedParameterById
 amici::Model, 207
 setFixedParameterByName
 amici::Model, 208
 setFixedParameters
 amici::Model, 205
 setFixedParametersByIdRegex
 amici::Model, 207
 setFixedParametersByNameRegex
 amici::Model, 208
 setHflag
 amievent, 506
 setId
 amici::Solver, 472
 setInitialStateSensitivities
 amici::Model, 213
 setInitialStates
 amici::Model, 212
 setInternalSensitivityMethod
 amici::Solver, 462
 setInterpolationType
 amici::Solver, 458
 setJacTimesVecFn
 amici::Solver, 465
 setJacTimesVecFnB
 amici::Solver, 466
 setLinearMultistepMethod
 amici::Solver, 456
 setLinearSolver
 amici::Solver, 461
 setMaxNumSteps
 amici::Solver, 470
 setMaxNumStepsB
 amici::Solver, 471
 setMaxSteps
 amici::Solver, 454
 setMaxStepsBackwardProblem
 amici::Solver, 455
 setModelData
 amici, 28
 setNMaxEvent
 amici::Model, 202
 setName
 amici::ode_export::ODEExporter, 373
 setNewtonMaxLinearSteps
 amici::Solver, 442
 setNewtonMaxSteps
 amici::Solver, 440
 setNewtonPreequilibration
 amici::Solver, 441
 setNonlinearSolverIteration
 amici::Solver, 457
 setObservedData
 amici::ExpData, 120
 setObservedDataStdDev
 amici::ExpData, 123–125
 setObservedEvents
 amici::ExpData, 127, 128
 setObservedEventsStdDev
 amici::ExpData, 130–132
 setParameterById
 amici::Model, 231
 setParameterByName
 amici::Model, 232
 setParameterList
 amici::Model, 212
 setParameterScale
 amici::Model, 203
 setParameters
 amici::Model, 204
 setParametersByIdRegex
 amici::Model, 232
 setParametersByNameRegex
 amici::Model, 233
 setPaths
 amici::ode_export::ODEExporter, 372
 setQuadErrConB
 amici::Solver, 469
 setReinitializeFixedParameterInitialStates
 amici::Model, 237
 setRelativeTolerance
 amici::Solver, 444
 setRelativeToleranceQuadratures
 amici::Solver, 449
 setRelativeToleranceSensi
 amici::Solver, 447
 setRelativeToleranceSteadyState
 amici::Solver, 451
 setRelativeToleranceSteadyStateSensi
 amici::Solver, 453
 setSStolerances
 amici::Solver, 467
 setSStolerancesB
 amici::Solver, 474
 setSensErrCon
 amici::Solver, 468
 setSensParams
 amici::Solver, 473
 setSensSStolerances
 amici::Solver, 468
 setSensitivityMethod
 amici::Solver, 439
 setSensitivityOrder
 amici::Solver, 443
 setSolverOptions
 amici, 29
 setSparseJacFn
 amici::Solver, 465
 setSparseJacFnB

amici::Solver, 466
setStabLimDet
 amici::Solver, 471
setStabLimDetB
 amici::Solver, 472
setStabilityLimitFlag
 amici::Solver, 460
setStatelsNonNegative
 amici::Model, 210
setStateOrdering
 amici::Solver, 459
setSteadyStateSensitivityMode
 amici::Model, 237
setStopTime
 amici::Solver, 437
setSuppressAlg
 amici::Solver, 473
setT0
 amici::Model, 214
setTimepoints
 amici::ExpData, 118
 amici::Model, 209
setUserData
 amici::Solver, 469
setUserDataB
 amici::Solver, 470
setValueById
 amici, 72
setValueByIdRegex
 amici, 73
setup
 amici::Solver, 429
setupAMIB
 amici::Solver, 430
SetupFailure, 423
 amici::SetupFailure, 424
setupReturnData
 amici, 30
seval
 amici, 50
Sigma_Y
 amidata, 503
Sigma_Z
 amidata, 504
SigmaY, 387
sigmay
 amici::Model, 271
 amici::ReturnData, 399
sigmaz
 amici::Model, 272
 amici::ReturnData, 397
sign
 amici, 60
sinteg
 amici, 51
sllh
 amici::ReturnData, 404
solve
 amici::Solver, 435
solveLinearSystem
 amici::NewtonSolver, 349
 amici::NewtonSolverDense, 353
 amici::NewtonSolverIterative, 356
 amici::NewtonSolverSparse, 360
solveB
 amici::Solver, 437
solveF
 amici::Solver, 436
Solver, 424
 amici::Solver, 428
solver
 amici::ForwardProblem, 145
solverMemory
 amici::Solver, 492
solverMemoryB
 amici::Solver, 492
solverWasCalled
 amici::Solver, 492
sparse_functions
 amici::ode_export, 79
sparseeq
 amici::ode_export::ODEModel, 381
sparseidx
 amimodel, 534
sparseidxB
 amimodel, 534
sparsesym
 amici::ode_export::ODEModel, 380
spbcg
 amici::Solver, 478
spbcgB
 amici::Solver, 478
spgmr
 amici::Solver, 477
spgmrB
 amici::Solver, 477
spline
 amici, 48, 60
spline.cpp, 569
spline_pos
 amici, 62
splineflag
 amimodel, 537
sptfqmr
 amici::Solver, 478
sptfqmrB
 amici::Solver, 479
sres
 amici::ReturnData, 400
srz
 amici::ReturnData, 398
ss
 amioption, 545
ss_atol
 amioption, 542
ss_rtol

amioption, 542
 ssigma
 amici::ReturnData, 399
 ssigmaz
 amici::ReturnData, 398
 State, 388
 stateIsNonNegative
 amici::Model, 279
 StateOrdering
 amici, 22
 status
 amici::ReturnData, 405
 stau
 amici::Model, 277
 SteadyStateSensitivityMode
 amici, 22
 steadyStateSensitivityMode
 amici::Model, 280
 SteadystateProblem, 493
 amici::SteadystateProblem, 494
 stldet
 amioption, 544
 storeBacktrace
 amici::AmiException, 87
 strsym
 amifun, 513
 strsym_old
 amifun, 513
 sx
 amici::ReturnData, 399
 sx0
 amici::ReturnData, 404
 amioption, 545
 sx0data
 amici::Model, 279
 sy
 amici::ReturnData, 399
 sym
 amici::ode_export::ODEModel, 379
 amifun, 513
 amimodel, 528
 sym_noopt
 amifun, 513
 symNames
 amici::ode_export::ODEModel, 385
 symbol_to_type
 amici::ode_export, 79
 symbolic_functions.cpp, 569
 sz
 amici::ReturnData, 397
 t
 amici::Model, 210
 amici::NewtonSolver, 351
 amidata, 503
 t0
 amici::Model, 213
 amimodel, 530
 t_steadystate
 amici::ReturnData, 403
 TemplateAmici, 390
 time
 amici::IntegrationFailure, 148
 amici::IntegrationFailureB, 150
 trigger
 amievent, 506
 ts
 amici::ExpData, 137
 amici::Model, 279
 amici::ReturnData, 397
 tstart
 amici::Model, 280
 amioption, 543
 turnOffRootFinding
 amici::Solver, 438
 ubw
 amici::Model, 270
 amimodel, 533
 unscaleParameters
 amici, 33, 34
 unscaledParameters
 amici::Model, 278
 updateHeaviside
 amici::Model, 221
 updateHeavisideB
 amici::Model, 222
 updatemodelName
 amimodel, 519
 updateRHS
 amimodel, 519
 updateWrapPath
 amimodel, 520
 val
 amici::ode_export::ODEModel, 383
 var_in_function_signature
 amici::ode_export, 76
 varidx
 amised, 549
 varsym
 amised, 550
 w
 amici::Model, 277
 warnMsgIdAndTxt
 amici, 74
 what
 amici::AmiException, 86
 workBackwardProblem
 amici::BackwardProblem, 106
 workForwardProblem
 amici::ForwardProblem, 140
 workSteadyStateProblem
 amici::SteadystateProblem, 494
 wrap_path
 amimodel, 536
 wrapErrorHandlerFn

amici::Solver, 467
writeCcode
 amifun, 510
writeCcode_sensi
 amifun, 509
writeMatlabField0
 amici, 38
writeMatlabField1
 amici, 39
writeMatlabField2
 amici, 40
writeMatlabField3
 amici, 41
writeMatlabField4
 amici, 42
writeMcode
 amifun, 510
writeNewtonOutput
 amici::SteadystateProblem, 498
wrms_sensi_steadystate
 amici::ReturnData, 403
wrms_steadystate
 amici::ReturnData, 403
wtype
 amimodel, 530

x
 amici::NewtonSolver, 352
 amici::ReturnData, 398

x0
 amici::ReturnData, 403
 amioption, 545

x0data
 amici::Model, 279

x_pos_tmp
 amici::Model, 280

xdot
 amici::NewtonSolver, 351
 amici::ReturnData, 397

Y
 amidata, 503

y
 amici::ReturnData, 399

Z
 amidata, 504

z
 amici::ReturnData, 397
 amievent, 507

z2event
 amici::Model, 271
 amimodel, 537
 amioption, 546