

AMICI

Generated by Doxygen 1.8.14

## Contents

<b>1 About AMICI</b>	<b>2</b>
<b>2 License Conditions</b>	<b>2</b>
<b>3 How to contribute</b>	<b>3</b>
<b>4 Installation</b>	<b>3</b>
<b>5 Python Interface</b>	<b>6</b>
<b>6 MATLAB Interface</b>	<b>8</b>
<b>7 C++ Interface</b>	<b>13</b>
<b>8 FAQ</b>	<b>14</b>
<b>9 Namespace Documentation</b>	<b>14</b>
9.1 amici Namespace Reference . . . . .	14
9.2 amici.plotting Namespace Reference . . . . .	71
9.3 amici.sbml_import Namespace Reference . . . . .	72
<b>10 Class Documentation</b>	<b>76</b>
10.1 AmiException Class Reference . . . . .	76
10.2 AmiVector Class Reference . . . . .	80
10.3 AmiVectorArray Class Reference . . . . .	88
10.4 BackwardProblem Class Reference . . . . .	94
10.5 CvodeException Class Reference . . . . .	100
10.6 ExpData Class Reference . . . . .	101
10.7 ForwardProblem Class Reference . . . . .	128
10.8 IDAException Class Reference . . . . .	136
10.9 IntegrationFailure Class Reference . . . . .	137
10.10 IntegrationFailureB Class Reference . . . . .	139
10.11 Model Class Reference . . . . .	140
10.12 Model_DAE Class Reference . . . . .	270

10.13Model_ODE Class Reference . . . . .	299
10.14NewtonFailure Class Reference . . . . .	327
10.15NewtonSolver Class Reference . . . . .	329
10.16NewtonSolverDense Class Reference . . . . .	336
10.17NewtonSolverIterative Class Reference . . . . .	339
10.18NewtonSolverSparse Class Reference . . . . .	343
10.19ReturnData Class Reference . . . . .	345
10.20SBMLError Class Reference . . . . .	361
10.21SbmlImporter Class Reference . . . . .	362
10.22TemplateAmici Class Reference . . . . .	394
10.23SetupFailure Class Reference . . . . .	395
10.24Solver Class Reference . . . . .	396
10.25SteadystateProblem Class Reference . . . . .	457
10.26amidata Class Reference . . . . .	463
10.27amievent Class Reference . . . . .	468
10.28amifun Class Reference . . . . .	470
10.29amimodel Class Reference . . . . .	478
10.30amioption Class Reference . . . . .	503
10.31amised Class Reference . . . . .	511
10.32optsym Class Reference . . . . .	514
<b>11 File Documentation</b> . . . . .	<b>515</b>
11.1 am_and.m File Reference . . . . .	515
11.2 am_eq.m File Reference . . . . .	516
11.3 am_ge.m File Reference . . . . .	517
11.4 am_gt.m File Reference . . . . .	518
11.5 am_if.m File Reference . . . . .	519
11.6 am_le.m File Reference . . . . .	520
11.7 am_lt.m File Reference . . . . .	521
11.8 am_max.m File Reference . . . . .	522
11.9 am_min.m File Reference . . . . .	523
11.10am_or.m File Reference . . . . .	524
11.11am_piecewise.m File Reference . . . . .	525
11.12am_stepfun.m File Reference . . . . .	526
11.13am_xor.m File Reference . . . . .	527
11.14amici.cpp File Reference . . . . .	528
11.15amiwrap.m File Reference . . . . .	529
11.16cblas.cpp File Reference . . . . .	530
11.17interface_matlab.cpp File Reference . . . . .	531
11.18SBML2AMICI.m File Reference . . . . .	533
11.19spline.cpp File Reference . . . . .	534
11.20symbolic_functions.cpp File Reference . . . . .	534

## 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 as `amici.DoubleVector` to `amici.Model.setTimepoints`

```
model.setTimepoints(amici.DoubleVector(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

- [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)
- 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)
- 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[])
- 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 isNaN ()`
- `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 (rdata, sigma_y, sigma_z)`

*Convenience wrapper for `ExpData` constructor.*
- `def runAmiciSimulations (model, solver, edata_list)`

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

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

*local helper function to set parameters*
- `int setValueByldRegex (std::vector< std::string > const &ids, std::vector< realtype > &values, realtype value, std::string const &regex, 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]

CVODES/IDAS linear solvers

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

#### Returns

`void`

Definition at line 130 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 <b>Type:</b> char
in	<i>format</i>	string with error message printf-style format
	...	arguments to be formatted

##### Returns

void

Definition at line 151 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

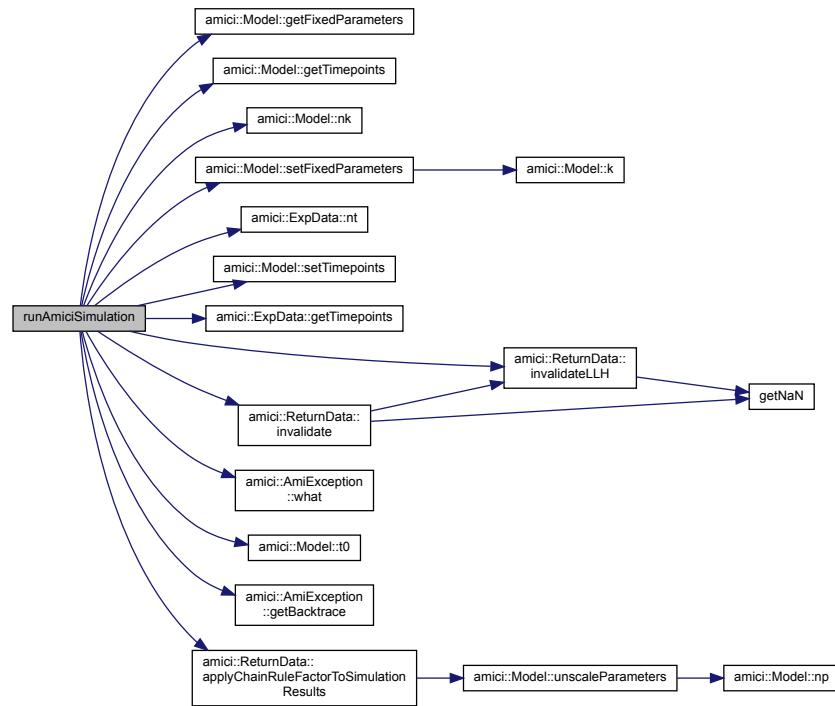
<i>solver</i>	<a href="#">Solver</a> instance
<i>edata</i>	pointer to experimental data object
<i>model</i>	model specification object

##### Returns

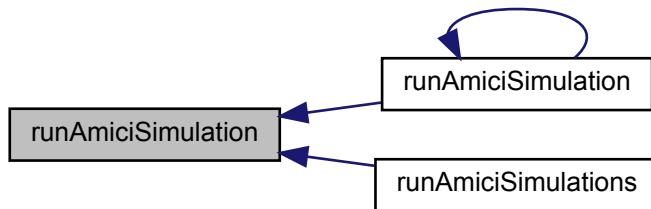
rdata pointer to return data object

Definition at line 59 of file amici.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.1.4.4 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  $y = \alpha \cdot A \cdot x + \beta \cdot y$  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  $y = \alpha \cdot A \cdot x + \beta \cdot y$  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.

### 9.1.4.5 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  $C = \alpha * A * B + \beta * C$  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  $C = \alpha * A * B + \beta * C$  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)

**Parameters**

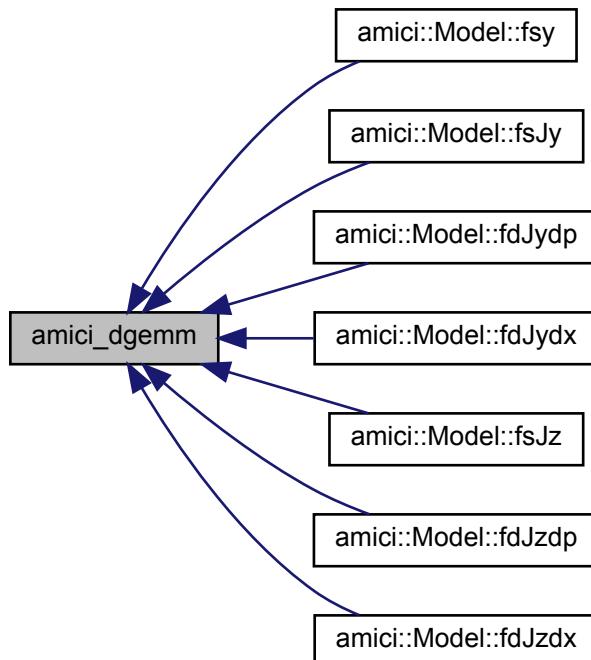
<i>beta</i>	coefficient beta
<i>C</i>	matrix C
<i>ldc</i>	leading dimension of C (m or n)

**Returns**

void

Definition at line 44 of file `cblas.cpp`.

Here is the caller graph for this function:

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

Definition at line 90 of file cblas.cpp.

**9.1.4.7 setModelData()**

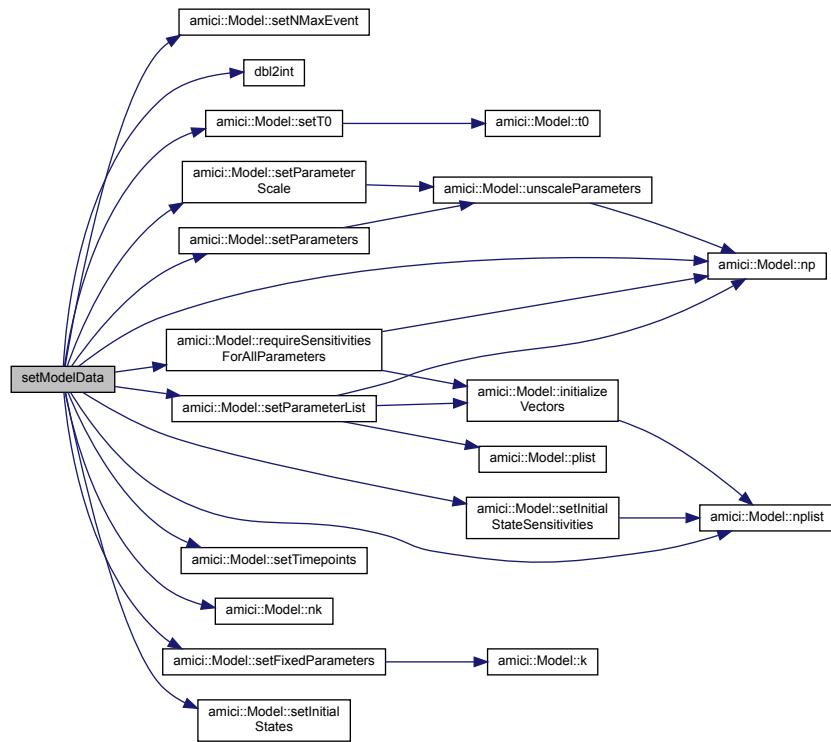
```
void setModelData (
    const mxArray * prhs[],
    int nrhs,
    Model & model )
```

**Parameters**

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

Definition at line 381 of file interface\_matlab.cpp.

Here is the call graph for this function:



#### 9.1.4.8 setSolverOptions()

```

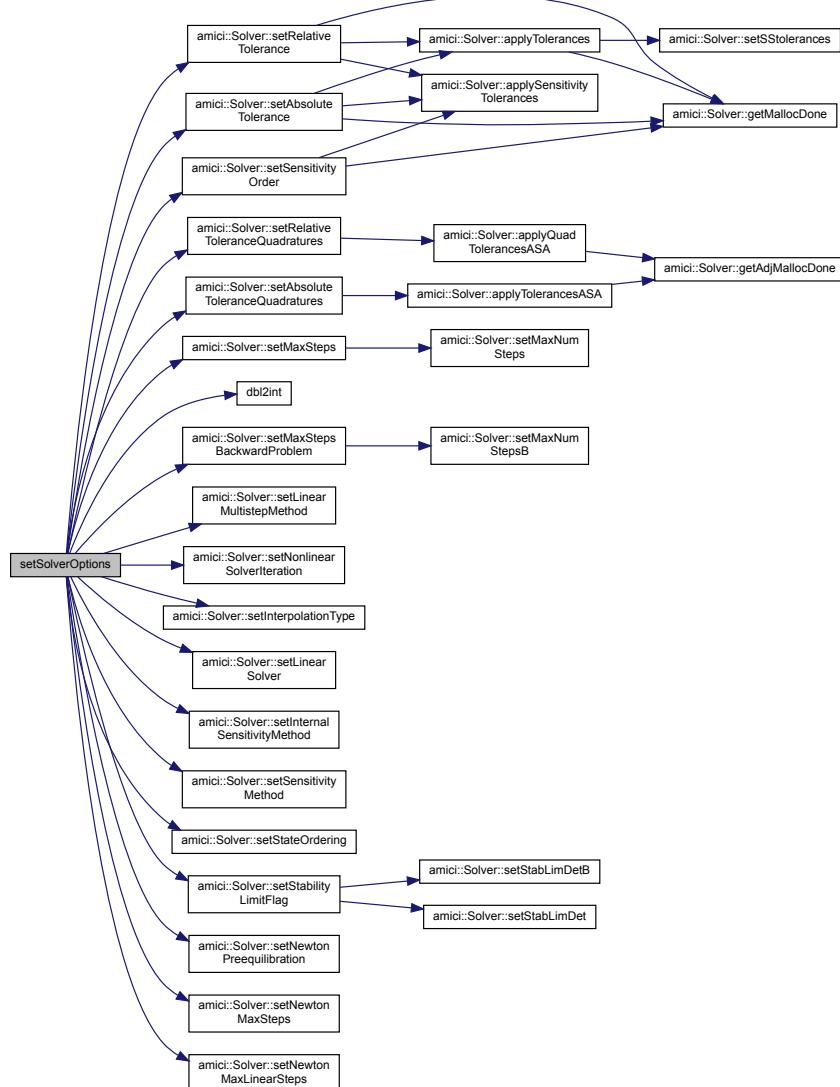
void setSolverOptions (
    const mxArray * prhs[],
    int nrhs,
    Solver & solver )
  
```

##### Parameters

in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> 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:



#### 9.1.4.9 setupReturnData()

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

setupReturnData initialises the return data struct

##### Parameters

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

**Returns**

rdata: return data struct

**Type:** \*ReturnData

**9.1.4.10 expDataFromMatlabCall()**

```
std::unique_ptr< ExpData > expDataFromMatlabCall (
```

const mxArray * <i>prhs</i> [],	<i>model</i> & <i>model</i> )
---------------------------------	-------------------------------

expDataFromMatlabCall initialises the experimental data struct

**Parameters**

in	<i>prhs</i>	user input <b>Type:</b> *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**

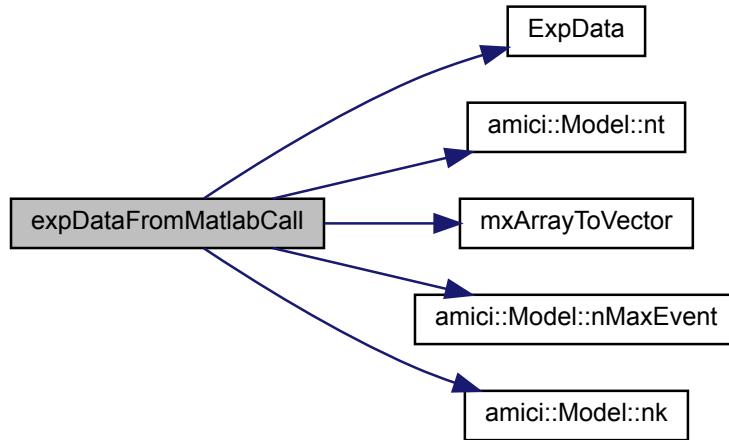
<i>prhs</i>	pointer to the array of input arguments
<i>model</i>	pointer to the model object, this is necessary to perform dimension checks

**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:



#### 9.1.4.11 `checkFinite()`

```

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

Checks the values in an array for NaNs and Infs

##### Parameters

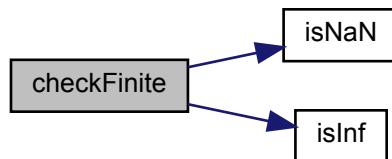
<code>N</code>	number of elements in array
<code>array</code>	array
<code>fun</code>	name of calling function

**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:

**9.1.4.12 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 1282 of file model.cpp.

#### 9.1.4.13 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

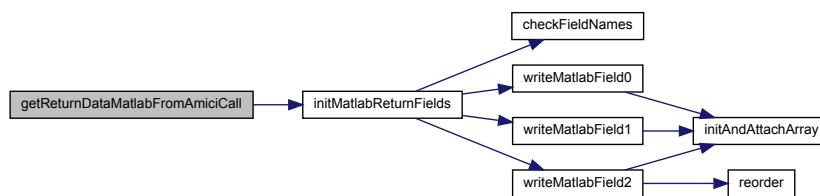
<code>rdata</code>	ReturnDataObject
--------------------	------------------

##### 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:



#### 9.1.4.14 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**

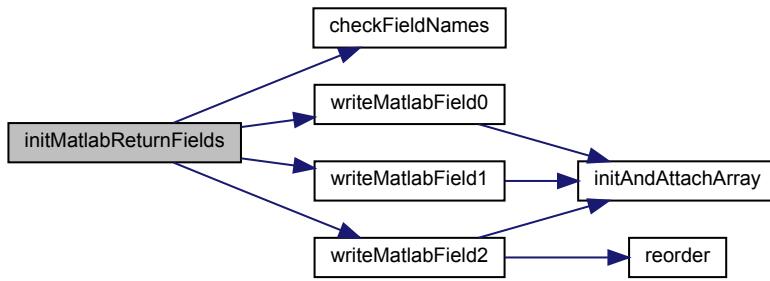
<code>rdata</code>	ReturnDataObject
--------------------	------------------

**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:

**9.1.4.15 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**

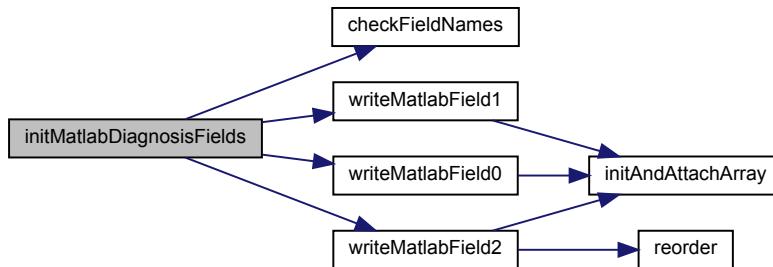
<i>rdata</i>	ReturnDataObject
--------------	------------------

**Returns**

Diagnosis mxArray

Definition at line 116 of file returndata\_matlab.cpp.

Here is the call graph for this function:

**9.1.4.16 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

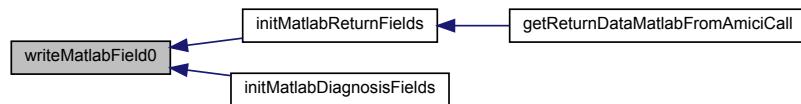
<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field

Definition at line 176 of file returndata\_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.1.4.17 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**

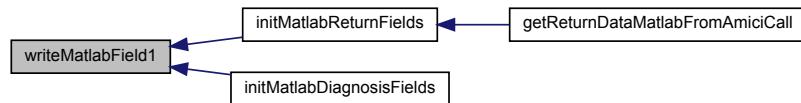
<i>matlabStruct</i>	pointer of the field to which the vector will be attached
<i>fieldName</i>	Name of the field to which the vector will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of elements in the vector

Definition at line 194 of file returndata\_matlab.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.1.4.18 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)

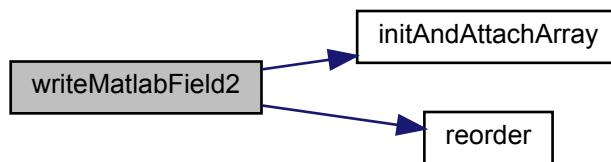
initialise matrix, attach to the field and write data

#### Parameters

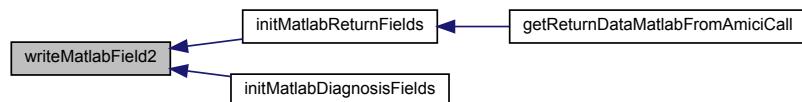
<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>fieldData</i>	Data which will be stored in the field
<i>dim0</i>	Number of rows in the tensor
<i>dim1</i>	Number of columns in the tensor
<i>perm</i>	reordering of dimensions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

Definition at line 216 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 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

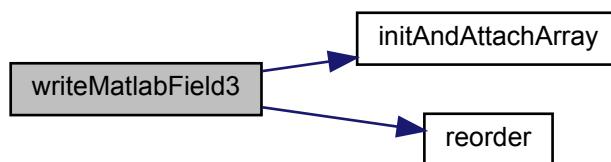
initialise 3D tensor, attach to the field and write data

**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 249 of file returndata\_matlab.cpp.

Here is the call graph for this function:

**9.1.4.20 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

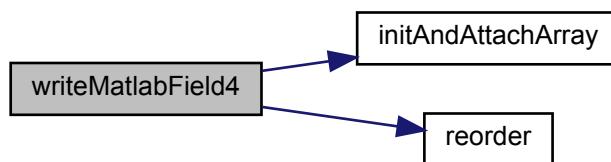
initialise 4D tensor, attach to the field and write data

**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 285 of file `returndata_matlab.cpp`.

Here is the call graph for this function:



#### 9.1.4.21 `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**

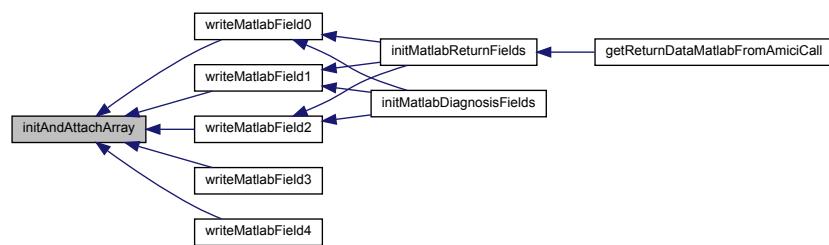
<i>matlabStruct</i>	Pointer to the matlab structure
<i>fieldName</i>	Name of the field to which the tensor will be attached
<i>dim</i>	vector of field dimensions

**Returns**

Pointer to field data

Definition at line 323 of file *returndata\_matlab.cpp*.

Here is the caller graph for this function:

**9.1.4.22 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>

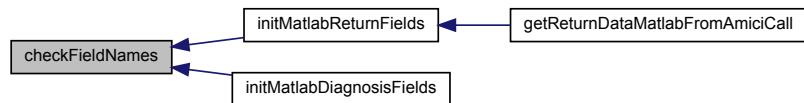
checks whether fieldNames was properly allocated

#### Parameters

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

Definition at line 344 of file returndata\_matlab.cpp.

Here is the caller graph for this function:



#### 9.1.4.23 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

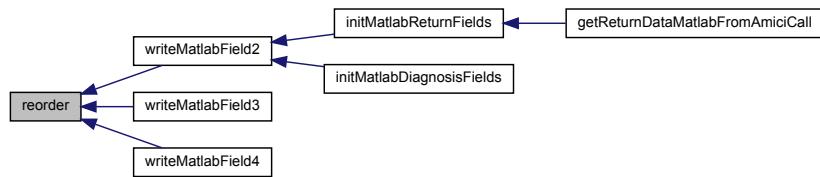
<i>input</i>	unordered vector
<i>order</i>	dimension permutation

#### Returns

Reordered vector

Definition at line 357 of file returndata\_matlab.cpp.

Here is the caller graph for this function:



#### 9.1.4.24 `serializeToChar()`

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

Serialize object to char array

##### Parameters

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

##### Returns

The object serialized as char

Definition at line 161 of file serialization.h.

#### 9.1.4.25 `deserializeFromChar()`

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

Deserialize object that has been serialized using `serializeToChar`

##### Parameters

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

##### Returns

The deserialized object

Definition at line 190 of file serialization.h.

#### 9.1.4.26 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 211 of file serialization.h.

#### 9.1.4.27 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 232 of file serialization.h.

#### 9.1.4.28 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 254 of file serialization.h.

**9.1.4.29 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.

**9.1.4.30 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**

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

## Return values

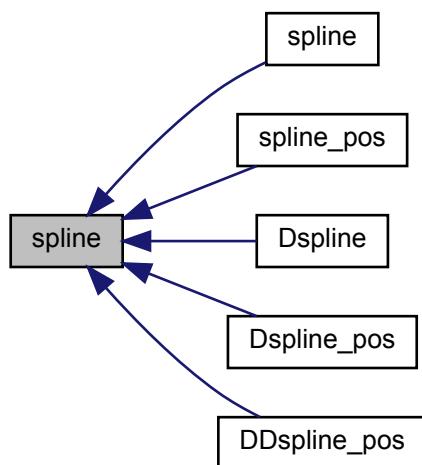
0	normal return
1	less than two data points; cannot interpolate
2	x[] are not in ascending order

## 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 .. 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:



### 9.1.4.31 seval()

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

$S(x) = y[i] + b[i] * w + c[i] * w^{**2} + d[i] * w^{**3}$  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 <a href="#">spline()</a> .
in	<i>c</i>	array of spline coefficients computed by <a href="#">spline()</a> .
in	<i>d</i>	array of spline coefficients computed by <a href="#">spline()</a> .

#### Returns

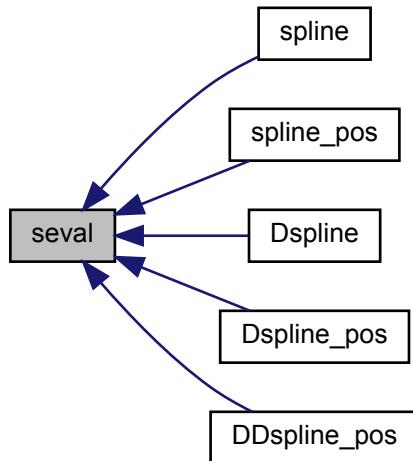
the value of the spline function at *u*

#### 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:



#### 9.1.4.32 sinteg()

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

Integrate the cubic spline function

$$S(xx) = y[i] + b[i] * w + c[i] * w^{**2} + d[i] * w^{**3} \text{ where } w = u - x[i] \text{ and } x[i] \leq u \leq x[i+1]$$

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 ( $n \geq 2$ )
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 <a href="#">spline()</a> .
in	<i>c</i>	array of spline coefficients computed by <a href="#">spline()</a> .
in	<i>d</i>	array of spline coefficients computed by <a href="#">spline()</a> .

**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.

**9.1.4.33 log()**

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

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

**Parameters**

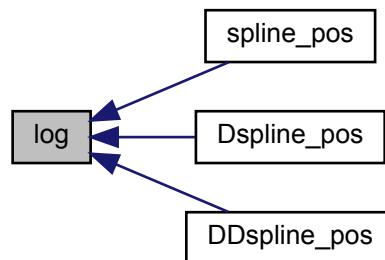
x	argument
---	----------

**Returns**

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

Definition at line 62 of file symbolic\_functions.cpp.

Here is the caller graph for this function:

**9.1.4.34 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.

**9.1.4.35 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.

**9.1.4.36 min()**

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

c implementation of matlab function min

**Parameters**

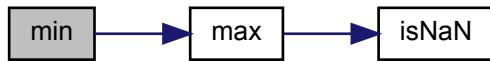
a	value1
b	value2
c	bogus parameter to ensure correct parsing as a function

**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:



#### 9.1.4.37 Dmin()

```
double Dmin (
    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 191 of file symbolic\_functions.cpp.

Here is the call graph for this function:



## 9.1.4.38 max()

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

c implementation of matlab function max

## Parameters

a	value1
b	value2
c	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:



## 9.1.4.39 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:

**9.1.4.40 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.

**9.1.4.41 isNaN()**

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

C++ interface to the isNaN function

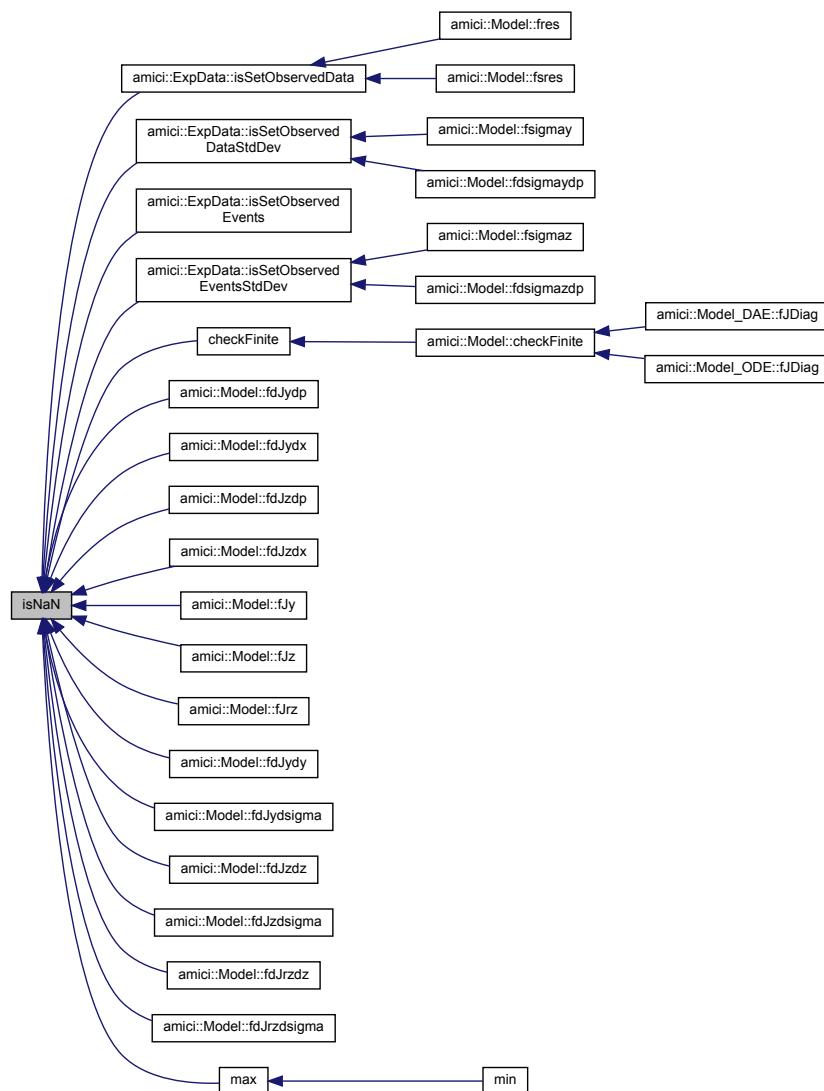
**Parameters**

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

**Returns**isnan(*what*)

Definition at line 35 of file symbolic\_functions.cpp.

Here is the caller graph for this function:

**9.1.4.42 isInf()**

```

int isInf (
    double what )

```

C++ interface to the isinf function

**Parameters**

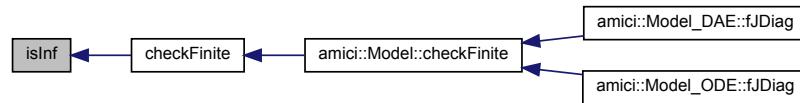
<i>what</i>	<i>argument</i>
-------------	-----------------

**Returns**

`isnan(what)`

Definition at line 44 of file `symbolic_functions.cpp`.

Here is the caller graph for this function:

**9.1.4.43 getNaN()**

```
double getNaN ( )
```

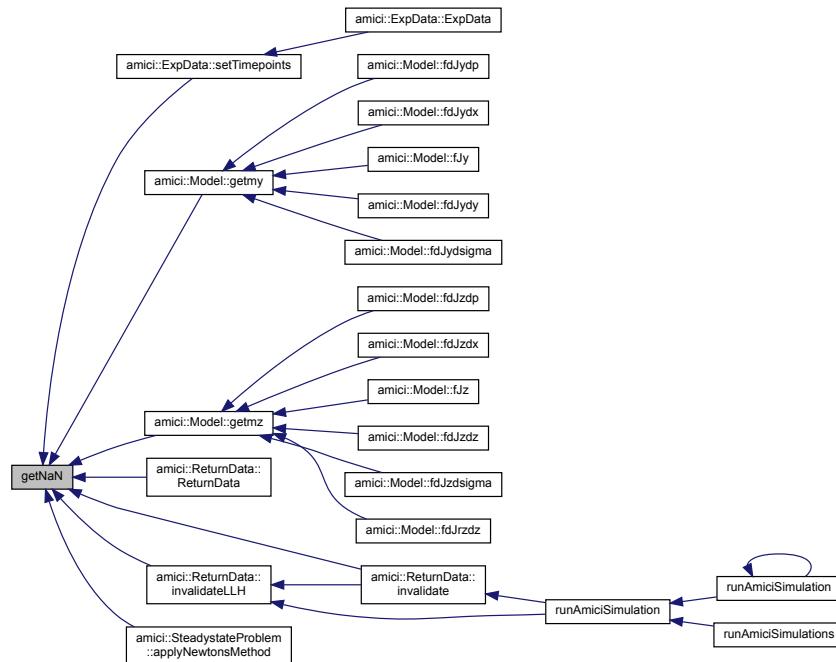
function returning nan

**Returns**

`Nan`

Definition at line 52 of file `symbolic_functions.cpp`.

Here is the caller graph for this function:



#### 9.1.4.44 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.

#### 9.1.4.45 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

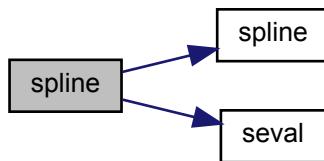
t	point at which the spline should be evaluated
num	number of spline nodes

**Returns**

```
spline(t)
```

Definition at line 222 of file symbolic\_functions.cpp.

Here is the call graph for this function:

**9.1.4.46 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**

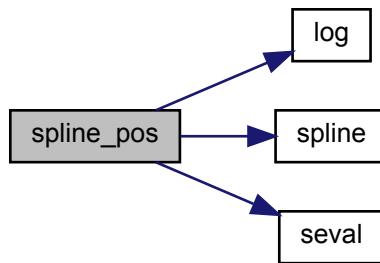
<code>t</code>	point at which the spline should be evaluated
<code>num</code>	number of spline nodes

**Returns**

```
spline(t)
```

Definition at line 273 of file symbolic\_functions.cpp.

Here is the call graph for this function:

**9.1.4.47 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**

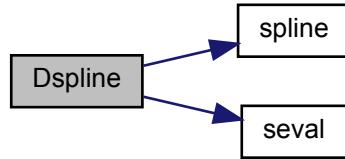
<i>id</i>	index of node to which the derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

**Returns**

```
dsplinedp(t)
```

Definition at line 326 of file symbolic\_functions.cpp.

Here is the call graph for this function:



#### 9.1.4.48 Dspline\_pos()

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

derivation of an exponentiated spline function, takes variable argument pairs ( $t_i, p_i$ ) with  $t_i$ : location of node i and  $p_i$ : 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

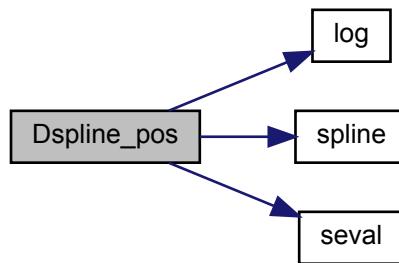
<code>id</code>	index of node to which the derivative of the corresponding spline coefficient should be computed
<code>t</code>	point at which the spline should be evaluated
<code>num</code>	number of spline nodes

##### Returns

`dsplinedp(t)`

Definition at line 382 of file `symbolic_functions.cpp`.

Here is the call graph for this function:



#### 9.1.4.49 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`.

#### 9.1.4.50 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  $t_i$ : location of node i and  $p_i$ : 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

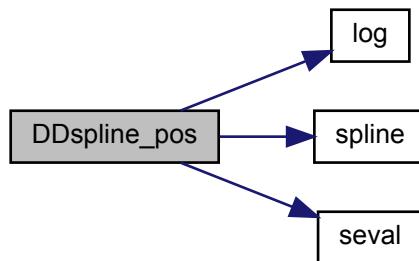
<code>id1</code>	index of node to which the first derivative of the corresponding spline coefficient should be computed
<code>id2</code>	index of node to which the second derivative of the corresponding spline coefficient should be computed
<code>t</code>	point at which the spline should be evaluated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

#### Returns

`ddspline(t)`

Definition at line 468 of file `symbolic_functions.cpp`.

Here is the call graph for this function:



#### 9.1.4.51 runAmiciSimulation() [2/2]

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

#### Parameters

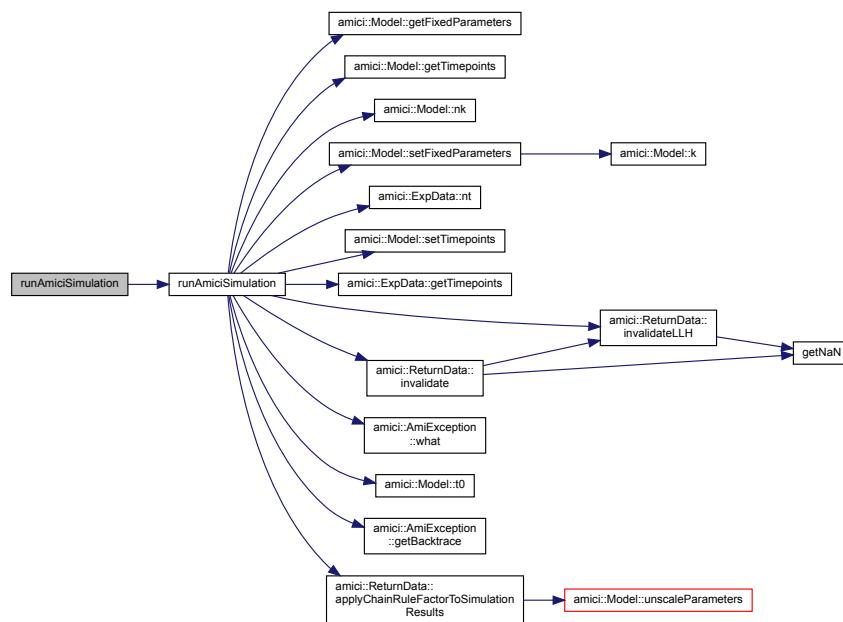
<code>model</code>	<a href="#">Model</a> instance
<code>solver</code>	<a href="#">Solver</a> instance, must be generated from <a href="#">Model.getSolver()</a>
<code>edata</code>	<a href="#">ExpData</a> instance (optional)

**Returns**

[ReturnData](#) object with simulation results

Definition at line 87 of file `__init__.py`.

Here is the call graph for this function:



Here is the caller graph for this function:

**9.1.4.52 ExpData()**

```

def amici.ExpData (
    rdata,
    sigma_y,
    sigma_z )
  
```

**Parameters**

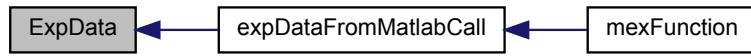
<i>rdata</i>	rdataToNumPyArrays output
<i>sigma<sub>y</sub></i>	standard deviation for ObservableData
<i>sigma<sub>z</sub></i>	standard deviation for EventData

**Returns**

[ExpData](#) Instance

Definition at line 107 of file `__init__.py`.

Here is the caller graph for this function:

**9.1.4.53 runAmiciSimulations()**

```
def amici.runAmiciSimulations (
    model,
    solver,
    edata_list )
```

**Parameters**

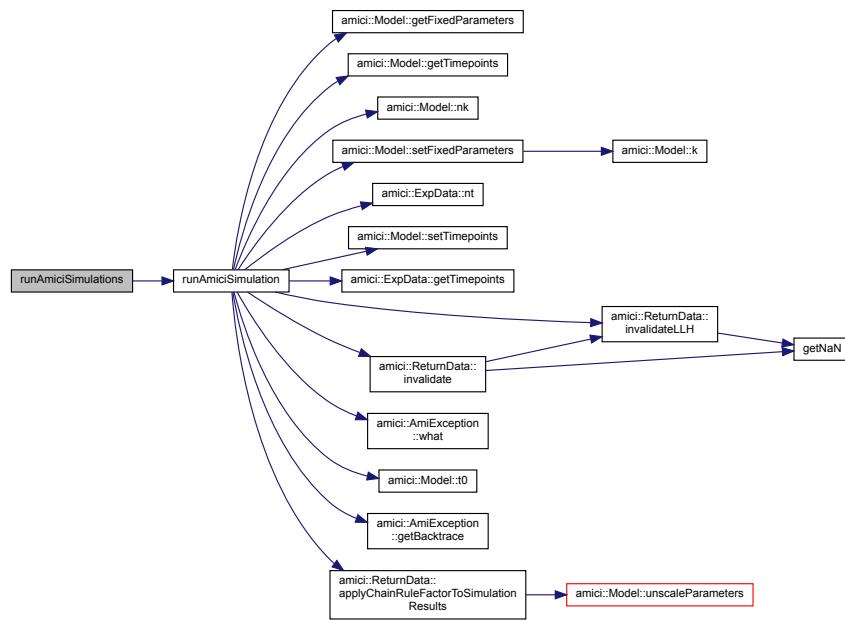
<i>model</i>	<a href="#">Model</a> instance
<i>solver</i>	<a href="#">Solver</a> instance, must be generated from <a href="#">Model.getSolver()</a>
<i>edata_list</i>	list of <a href="#">ExpData</a> instances

**Returns**

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

Definition at line 124 of file `__init__.py`.

Here is the call graph for this function:



#### 9.1.4.54 dbl2int()

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

conversion from double to int with checking for loss of data

##### Parameters

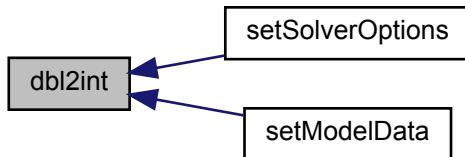
x	input
---	-------

**Returns**

`int_x` casted value

Definition at line 298 of file `interface_matlab.cpp`.

Here is the caller graph for this function:

**9.1.4.55 amici\_blasCBlasTransToBlasTrans()**

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

`amici_blasCBlasTransToBlasTrans` translates AMICI\_BLAS\_TRANSPOSE values to CBlas readable strings

**Parameters**

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

**Returns**

`cblastrans` CBlas readable CHAR indicating transposition and complex conjugation

Definition at line 52 of file `interface_matlab.cpp`.

**9.1.4.56 mxArrayToVector()**

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

conversion from mxArray to vector<realtype>

**Parameters**

<code>array</code>	Matlab array to create vector from
<code>length</code>	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:

**9.1.4.57 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**

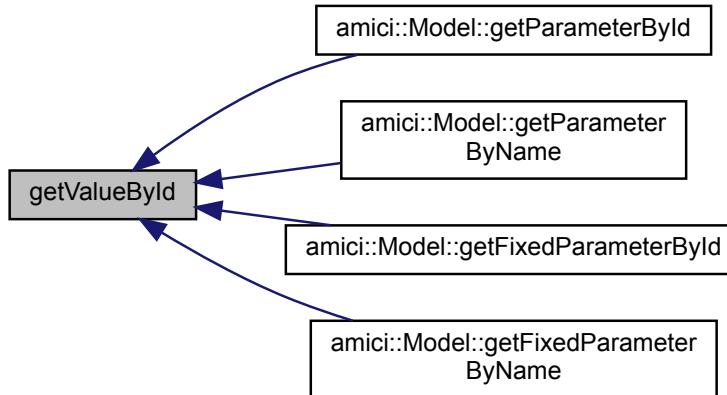
<i>ids</i>	vector of name/ids of (fixed)Parameters
<i>values</i>	values of the (fixed)Parameters
<i>id</i>	name/id to look for in the vector
<i>variable_name</i>	string indicating what variable we are lookin at
<i>id_name</i>	string indicating whether name or id was specified

**Returns**

value of the selected parameter

Definition at line 322 of file `model.cpp`.

Here is the caller graph for this function:



#### 9.1.4.58 setValueById()

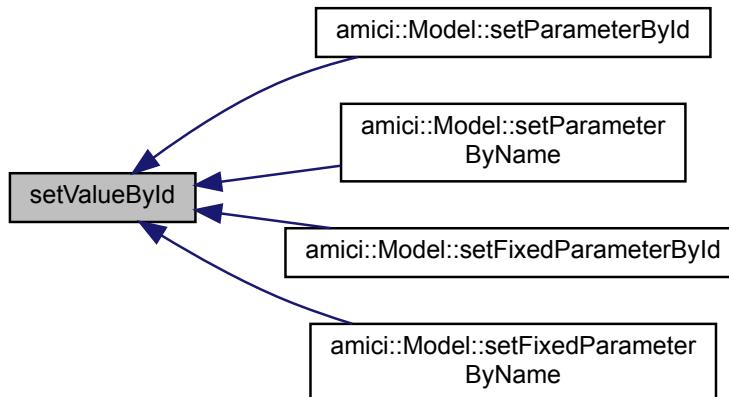
```
void amici::setValueById (
    std::vector< std::string > const & ids,
    std::vector< realltype > & values,
    realltype value,
    std::string const & id,
    const char * variable_name,
    const char * id_name )
```

##### 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 340 of file model.cpp.

Here is the caller graph for this function:



#### 9.1.4.59 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

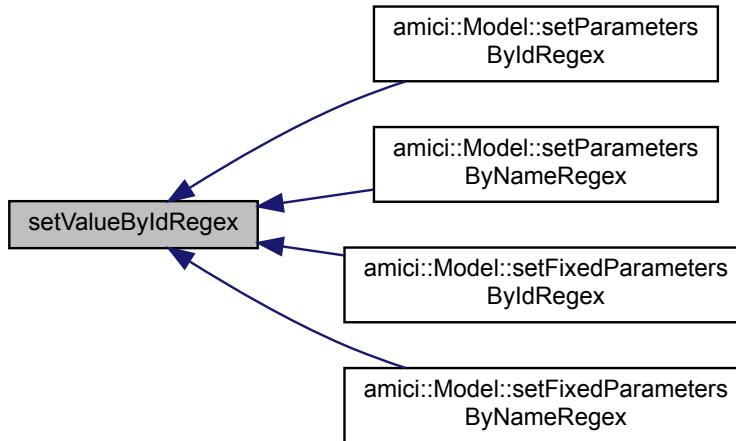
<i>ids</i>	vector of names/ids of (fixed)Parameters
<i>values</i>	values of the (fixed)Parameters
<i>value</i>	for the selected parameter
<i>regex</i>	string according to which names/ids are to be matched
<i>variable_name</i>	string indicating what variable we are lookin at
<i>id_name</i>	string indicating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

##### Returns

number of matched names/ids

Definition at line 359 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 45 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.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.*

### 9.2.1 Function Documentation

#### 9.2.1.1 plotStateTrajectories()

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

##### Parameters

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

##### Returns

Definition at line 17 of file plotting.py.

### 9.2.1.2 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`.

## 9.3 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).*
- class [TemplateAmici](#)  
*Template format used in AMICI (see `string.template` for more details).*

#### Functions

- def [applyTemplate](#) (sourceFile, targetFile, templateData)  
*Load source file, apply template substitution as provided in templateData and save as targetFile.*
- def [getSymbols](#) (prefix, length)  
*Get symbolic matrix with symbols prefix0..prefix(length-1).*
- def [getSymbolicDiagonal](#) (matrix)  
*Get symbolic matrix with diagonal of matrix matrix.*
- 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)*

### 9.3.1 Detailed Description

!/usr/bin/env python3

### 9.3.2 Function Documentation

#### 9.3.2.1 applyTemplate()

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

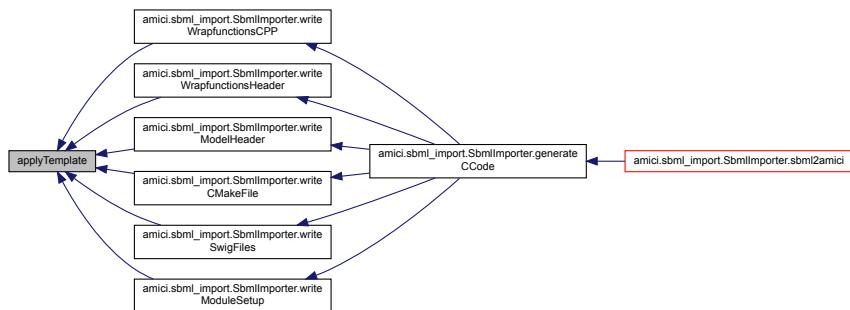
##### Parameters

<i>sourceFile</i>	relative or absolute path to template file
<i>targetFile</i>	relative or absolute path to output file
<i>templateData</i>	dictionary with template keywords to substitute (key is template variable without <a href="#">TemplateAmici.delimiter</a> )

##### Returns

Definition at line 1710 of file sbml\_import.py.

Here is the caller graph for this function:



#### 9.3.2.2 getSymbols()

```
def amici.sbml_import.getSymbols (
    prefix,
    length )
```

**Parameters**

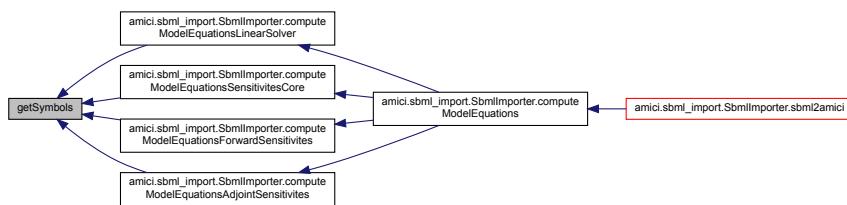
<i>prefix</i>	variable name
<i>length</i>	number of symbolic variables + 1

**Returns**

A symbolic matrix with symbols `prefix0..prefix(length-1)`

Definition at line 1729 of file `sbml_import.py`.

Here is the caller graph for this function:

**9.3.2.3 getSymbolicDiagonal()**

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

**Parameters**

<i>matrix</i>	Matrix from which to return the diagonal
---------------	--

**Returns**

A Symbolic matrix with the diagonal of `matrix`.

**Exceptions**

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

Definition at line 1744 of file `sbml_import.py`.

Here is the caller graph for this function:



### 9.3.2.4 getRuleVars()

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

#### Parameters

<code>rules</code>	list of rules
--------------------	---------------

#### Returns

Vector of free symbolic variables in the formulas all provided rules

Definition at line 1763 of file sbml\_import.py.

Here is the caller graph for this function:



### 9.3.2.5 assignmentRules2observables()

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

#### Parameters

<code>sbml_model</code>	an sbml Model instance
<code>filter_function</code>	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 1798 of file sbml\_import.py.

### 9.3.2.6 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 1827 of file sbml\_import.py.

**9.3.2.7 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 ( $\log(x, \text{basis})$ ) and libsbml ( $\log(\text{basis}, x)$ ).

**Parameters**

<i>x</i>	string to replace
----------	-------------------

**Returns**

string with replaced 'log's

Definition at line 1871 of file sbml\_import.py.

Here is the caller graph for this function:



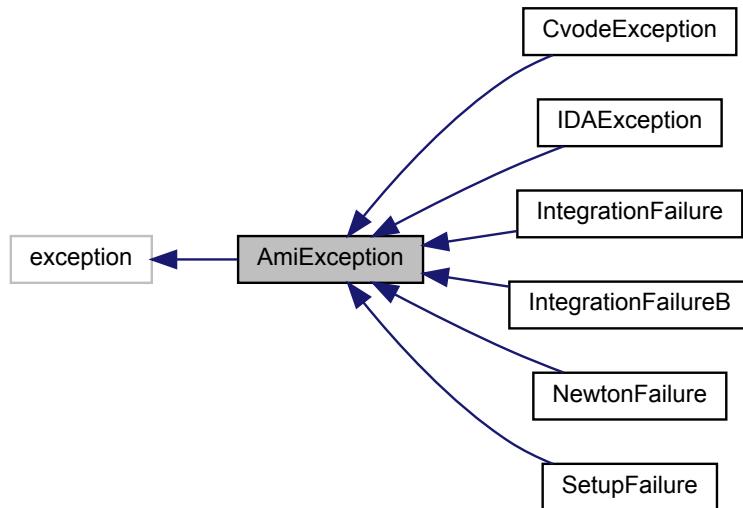
## 10 Class Documentation

### 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 ()
```

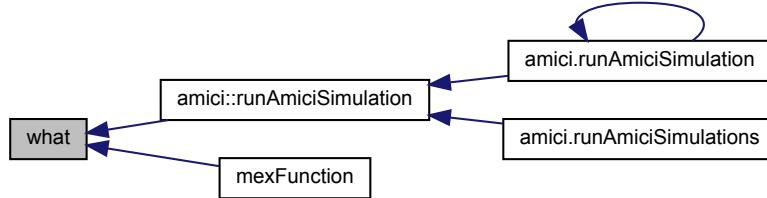
override of default error message function

**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
```

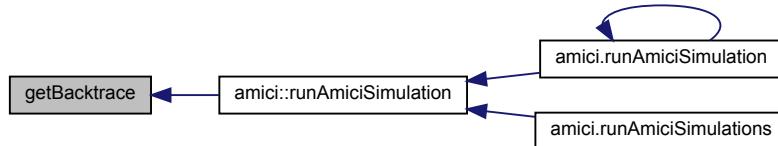
returns the stored backtrace

**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**

<i>nMaxFrames</i>	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= \(AmiVector &other\)](#)
- [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 & 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]

```
realtype* data ( )
```

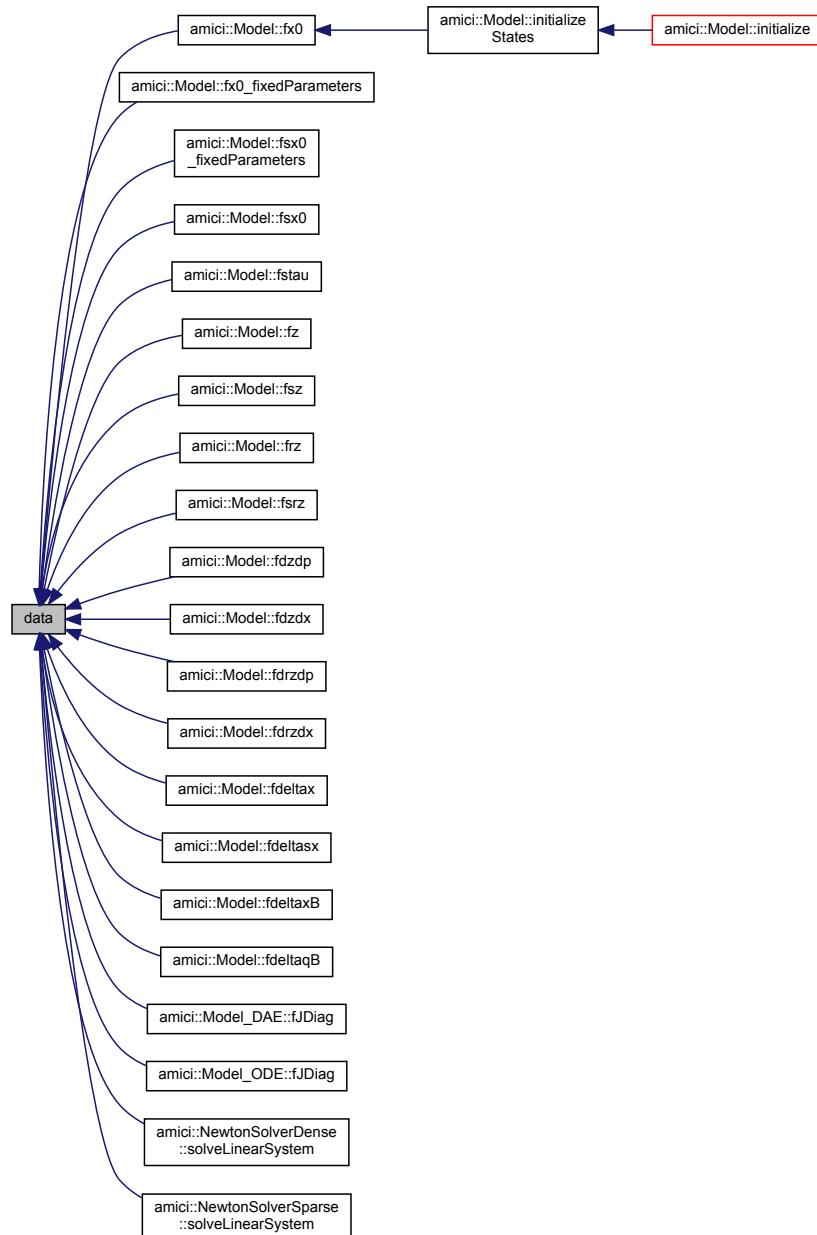
data accessor

**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
```

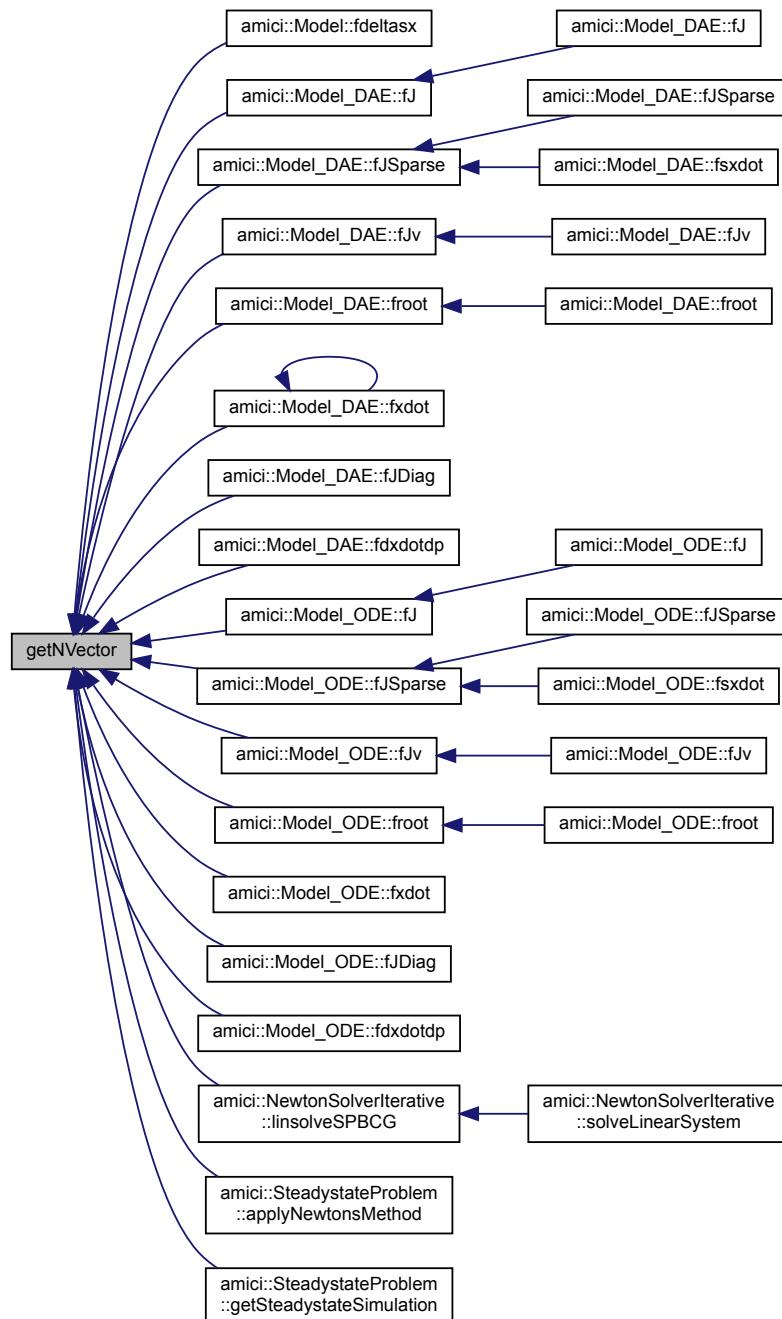
N\_Vector accessor

**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<realtyp> const& getVector( ) const
```

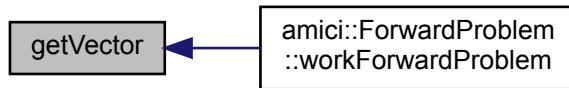
Vector accessor

**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
```

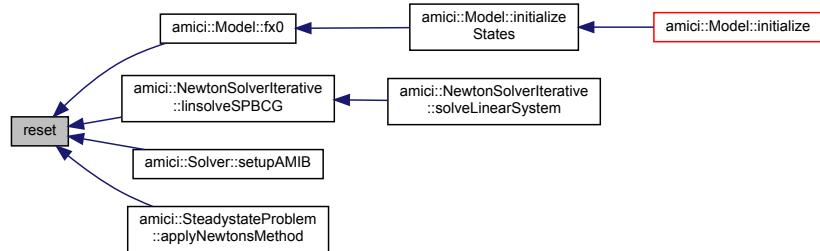
Definition at line 91 of file vector.h.

**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:



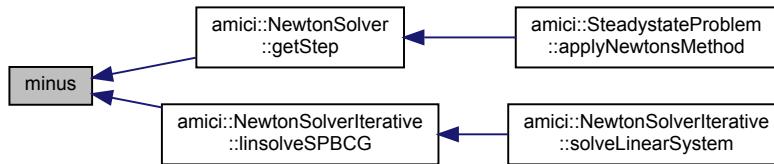
### 10.2.3.8 minus()

```
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

val	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**

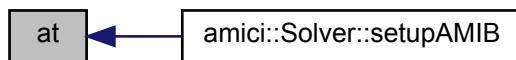
<i>pos</i>	index of element
------------	------------------

**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

<code>length_inner</code>	length of vectors
<code>length_outer</code>	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**

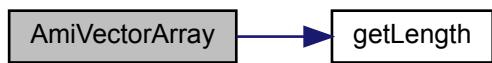
<i>vaold</i>	object to copy from
--------------	---------------------

**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**

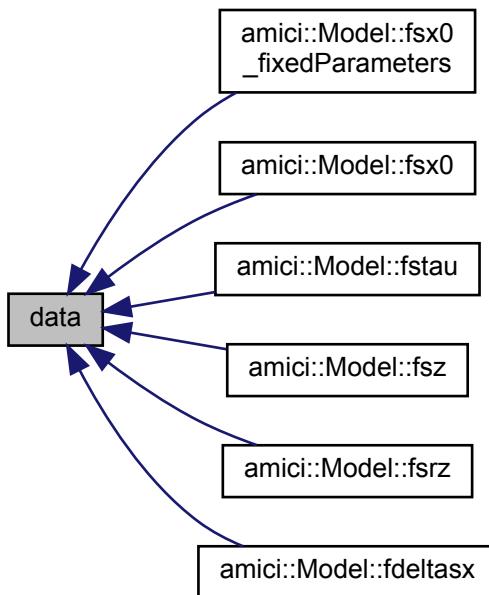
<i>pos</i>	index of <a href="#">AmiVector</a>
------------	------------------------------------

**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**

<i>pos</i>	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

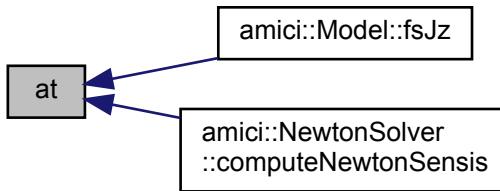
<i>ipos</i>	inner index in AmiVector
<i>jpos</i>	outer index in AmiVectorArray

#### 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 <a href="#">AmiVector</a>
------------	--

**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 <a href="#">AmiVector</a>
------------	------------------------------------

**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 <a href="#">AmiVector</a>
------------	------------------------------------

**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:



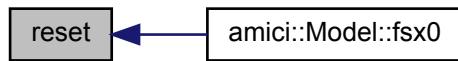
### 10.3.3.9 `reset()`

`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

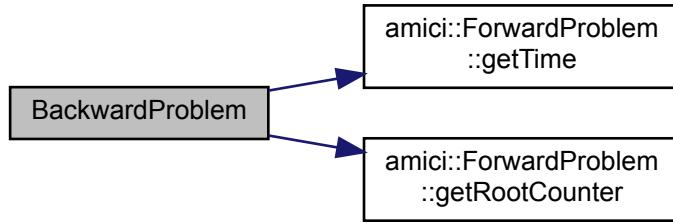
<i>fwd</i>	pointer to corresponding forward problem
------------	--

###### 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
```

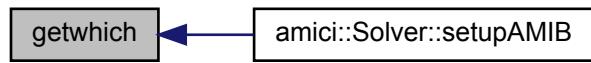
accessor for which

**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 ( )
```

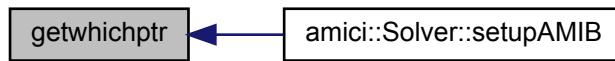
accessor for pointer to which

##### 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 ( )
```

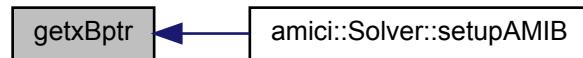
accessor for pointer to xB

##### Returns

&x<sub>B</sub>

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 ( )
```

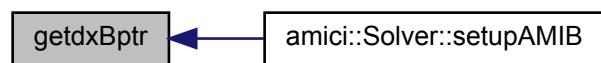
accessor for pointer to dxB

##### Returns

&dxB

Definition at line 67 of file backwardproblem.h.

Here is the caller graph for this function:



#### 10.4.3.8 `getdJydx()`

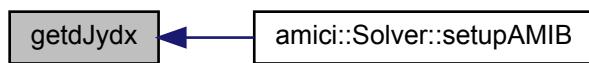
```
std::vector<realtype> const& getdJydx ( ) const
accessor for dJydx
```

##### Returns

`dJydx`

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

Here is the caller graph for this function:

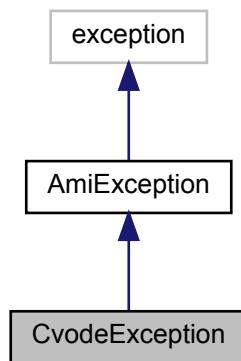


## 10.5 CvodeException Class Reference

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 \(int nytrue, int nztrue, int nmaxevent\)](#)
- [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 \(const ReturnData &rdata, std::vector< realtype > sigma\\_y, std::vector< realtype > sigma\\_z\)](#)
- [ExpData \(const ExpData &other\)](#)

*Copy constructor.*

  - [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

- const int `nytrue`
- const int `nztrue`
- const int `nmaxevent`
- 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

- 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 &amp; 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 (
    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.

## 10.6.2.3 ExpData() [3/8]

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

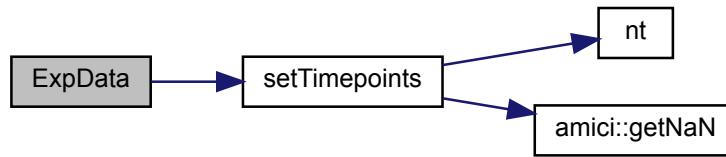
constructor that initializes timepoints from vectors

**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:



#### 10.6.2.4 ExpData() [4/8]

```
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 )
```

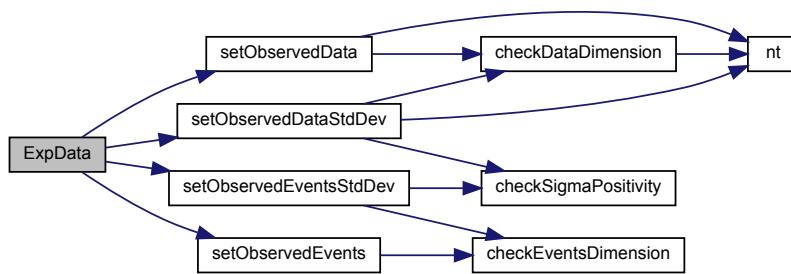
constructor that initializes timepoints and data from vectors

##### 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 eedata.cpp.

Here is the call graph for this function:



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

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

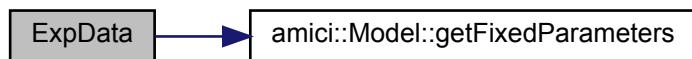
constructor that initializes with [Model](#)

##### Parameters

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

Definition at line 42 of file eedata.cpp.

Here is the call graph for this function:



#### 10.6.2.6 ExpData() [6/8]

```
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<sub>y</sub></i>	scalar standard deviations for all observables
<i>sigma<sub>z</sub></i>	scalar standard deviations for all event observables

Definition at line 56 of file edata.cpp.

**10.6.2.7 ExpData() [7/8]**

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

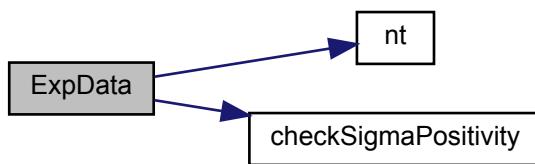
constructor that initializes with returnData, adds

**Parameters**

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

Definition at line 61 of file edata.cpp.

Here is the call graph for this function:

**10.6.2.8 ExpData() [8/8]**

```
ExpData (
    const ExpData & other )
```

**Parameters**

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

Definition at line 48 of file edata.cpp.

### 10.6.3 Member Function Documentation

#### 10.6.3.1 nt()

```
const int nt ( ) const
```

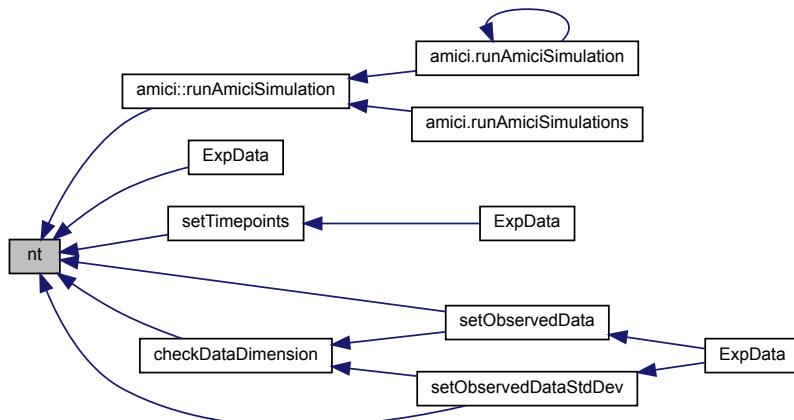
number of timepoints

**Returns**

number of timepoints

Definition at line 110 of file edata.cpp.

Here is the caller graph for this function:



#### 10.6.3.2 setTimepoints()

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

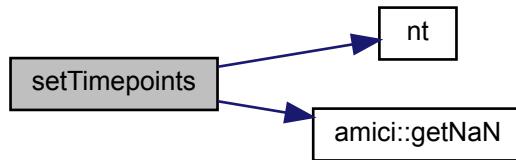
set function that copies data from input to [ExpData::ts](#)

**Parameters**

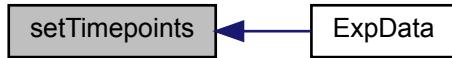
<i>ts</i>	timepoints
-----------	------------

Definition at line 97 of file eedata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.6.3.3 getTimepoints()**

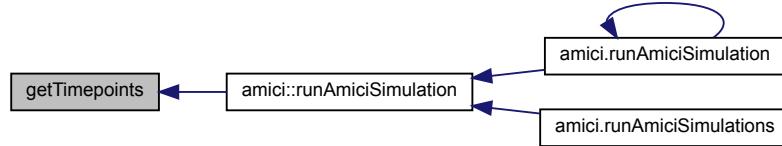
```
std::vector< realtypes > const & getTimepoints( ) const
get function that copies data from ExpData::ts to output
```

**Returns**

*ExpData::ts*

Definition at line 106 of file eedata.cpp.

Here is the caller graph for this function:



#### 10.6.3.4 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 114 of file edata.cpp.

#### 10.6.3.5 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 118 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.6.3.6 setObservedData() [2/2]

```
void setObservedData (
    const std::vector< realtype > & 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 127 of file edata.cpp.

Here is the call graph for this function:



#### 10.6.3.7 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**

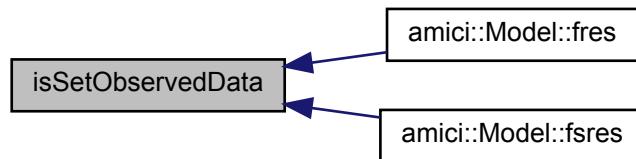
boolean specifying if data was set

Definition at line 135 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.6.3.8 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 139 of file edata.cpp.

**10.6.3.9 getObservedDataPtr()**

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

get function that returns a pointer to observed data at index

**Parameters**

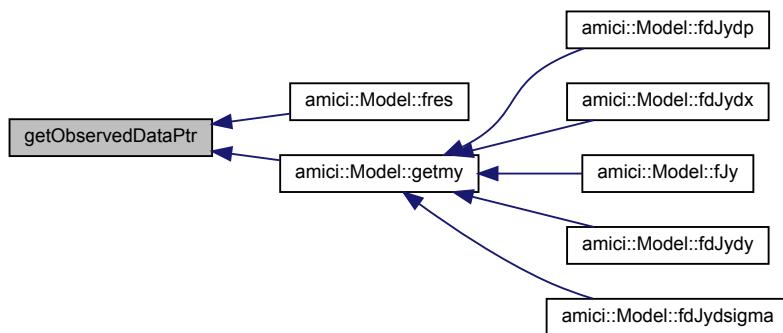
<i>it</i>	timepoint index
-----------	-----------------

**Returns**

pointer to observed data at index (dimension: nytrue)

Definition at line 143 of file edata.cpp.

Here is the caller graph for this function:

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

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

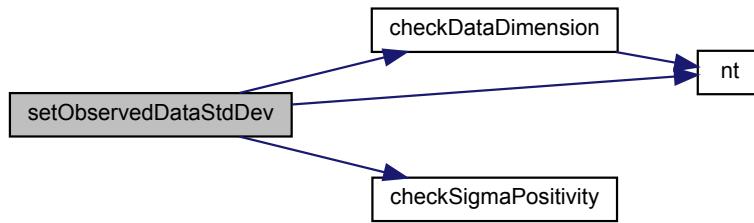
set function that copies data from input to [ExpData::observedDataStdDev](#)

**Parameters**

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

Definition at line 150 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.6.3.11 setObservedDataStdDev() [2/4]

```
void setObservedDataStdDev (const realltype stdDev )
```

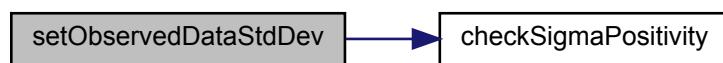
set function that sets all `ExpData::observedDataStdDev` to the input value

##### Parameters

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

Definition at line 160 of file `edata.cpp`.

Here is the call graph for this function:



### 10.6.3.12 setObservedDataStdDev() [3/4]

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

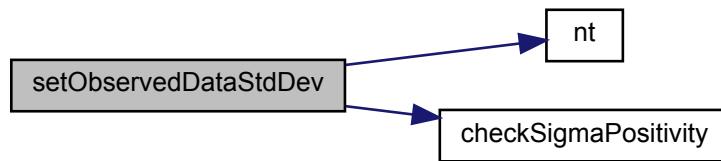
set function that copies standard deviation of observed data for specific observable

#### Parameters

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

Definition at line 165 of file edata.cpp.

Here is the call graph for this function:



### 10.6.3.13 setObservedDataStdDev() [4/4]

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

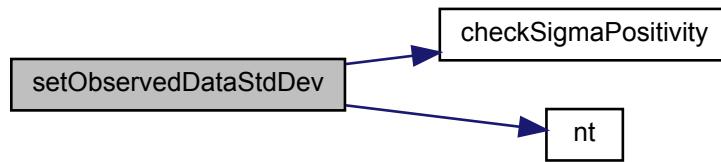
set function that sets all standard deviation of a specific observable to the input value

#### Parameters

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

Definition at line 174 of file edata.cpp.

Here is the call graph for this function:



#### 10.6.3.14 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

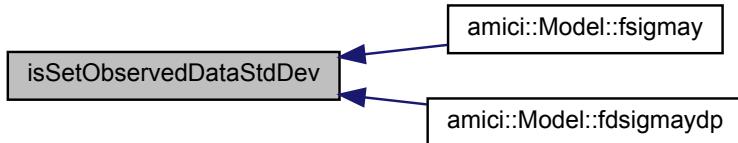
boolean specifying if standard deviation of data was set

Definition at line 180 of file eedata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.6.3.15 `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 184 of file `edata.cpp`.

#### 10.6.3.16 `getObservedDataStdDevPtr()`

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

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

##### Parameters

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

##### Returns

pointer to standard deviation of observed data at index

Definition at line 188 of file `edata.cpp`.

Here is the caller graph for this function:



#### 10.6.3.17 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 195 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.6.3.18 setObservedEvents() [2/2]

```
void setObservedEvents (
    const std::vector< realtypes > & 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 204 of file edata.cpp.

### 10.6.3.19 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 213 of file edata.cpp.

Here is the call graph for this function:



**10.6.3.20 getObservedEvents()**

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

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

**Returns**

observed event data

Definition at line 217 of file edata.cpp.

**10.6.3.21 getObservedEventsPtr()**

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

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

**Parameters**

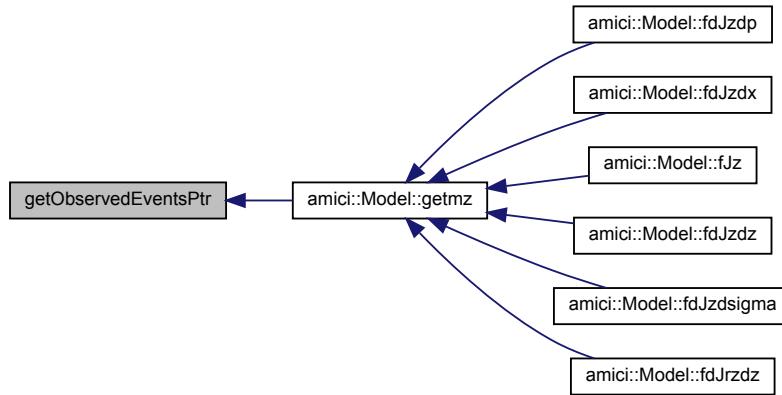
<i>ie</i>	event occurence
-----------	-----------------

**Returns**

pointer to observed event data at ieth occurence

Definition at line 221 of file edata.cpp.

Here is the caller graph for this function:



### 10.6.3.22 setObservedEventsStdDev() [1/4]

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

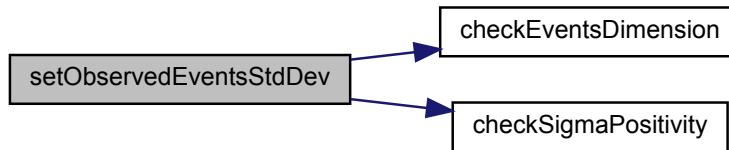
set function that copies data from input to [ExpData::observedEventsStdDev](#)

#### Parameters

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

Definition at line 228 of file eedata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.6.3.23 setObservedEventsStdDev() [2/4]

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

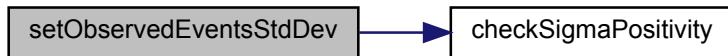
set function that sets all [ExpData::observedDataStdDev](#) to the input value

#### Parameters

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

Definition at line 238 of file eedata.cpp.

Here is the call graph for this function:



#### 10.6.3.24 setObservedEventsStdDev() [3/4]

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

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

##### Parameters

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

Definition at line 243 of file eedata.cpp.

Here is the call graph for this function:



#### 10.6.3.25 setObservedEventsStdDev() [4/4]

```
void setObservedEventsStdDev (
    const realtypes 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 252 of file edata.cpp.

Here is the call graph for this function:

**10.6.3.26 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**

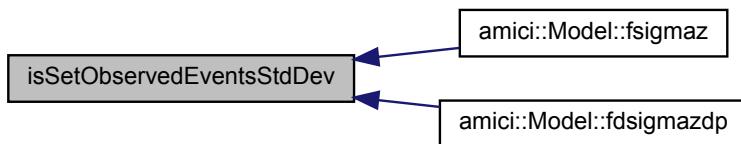
boolean specifying if standard deviation of event data was set

Definition at line 259 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.6.3.27 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 266 of file `edata.cpp`.

#### 10.6.3.28 getObservedEventsStdDevPtr()

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

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

##### Parameters

<code>ie</code>	event occurence
-----------------	-----------------

##### Returns

pointer to standard deviation of observed event data at ieth occurrence

Definition at line 270 of file `edata.cpp`.

Here is the caller graph for this function:



#### 10.6.3.29 checkDataDimension()

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

checker for dimensions of input observedData or observedDataStdDev

##### Parameters

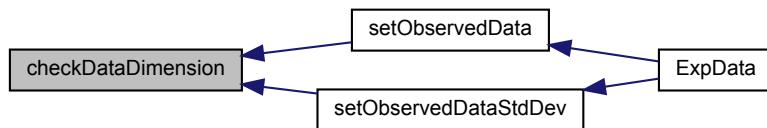
<i>input</i>	vector input to be checked
<i>fieldname</i>	name of the input

Definition at line 277 of file edata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.6.3.30 checkEventsDimension()

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

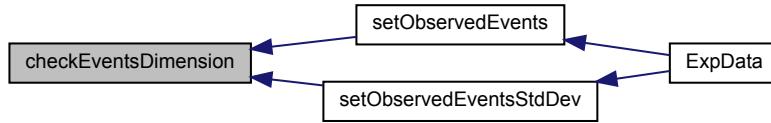
checker for dimensions of input observedEvents or observedEventsStdDev

**Parameters**

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

Definition at line 282 of file edata.cpp.

Here is the caller graph for this function:



## 10.6.3.31 checkSigmaPositivity() [1/2]

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

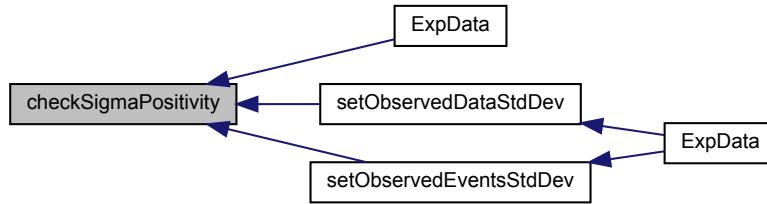
checks input vector of sigmas for not strictly positive values

**Parameters**

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

Definition at line 287 of file edata.cpp.

Here is the caller graph for this function:



### 10.6.3.32 `checkSigmaPositivity()` [2/2]

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

checks input scalar sigma for not strictly positive value

#### Parameters

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

Definition at line 292 of file `edata.cpp`.

## 10.6.4 Member Data Documentation

### 10.6.4.1 `nytrue`

`const int nytrue`

number of observables

Definition at line 309 of file `edata.h`.

### 10.6.4.2 `nztrue`

`const int nztrue`

number of event observables

Definition at line 311 of file `edata.h`.

#### 10.6.4.3 nmaxevent

```
const int nmaxevent
```

maximal number of event occurrences

Definition at line 313 of file edata.h.

#### 10.6.4.4 fixedParameters

```
std::vector<realtyp> fixedParameters
```

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

Definition at line 316 of file edata.h.

#### 10.6.4.5 fixedParametersPreequilibration

```
std::vector<realtyp> fixedParametersPreequilibration
```

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

Definition at line 319 of file edata.h.

#### 10.6.4.6 fixedParametersPresimulation

```
std::vector<realtyp> fixedParametersPresimulation
```

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

Definition at line 321 of file edata.h.

#### 10.6.4.7 ts

```
std::vector<realtyp> ts [protected]
```

observation timepoints (dimension: nt)

Definition at line 364 of file edata.h.

#### 10.6.4.8 observedData

```
std::vector<realtyp> observedData [protected]
```

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

Definition at line 367 of file edata.h.

#### 10.6.4.9 observedDataStdDev

```
std::vector<realtyp> observedDataStdDev [protected]
```

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

Definition at line 369 of file edata.h.

#### 10.6.4.10 observedEvents

```
std::vector<realtyp> observedEvents [protected]
```

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

Definition at line 372 of file edata.h.

#### 10.6.4.11 observedEventsStdDev

```
std::vector<realtyp> observedEventsStdDev [protected]
```

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

Definition at line 375 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 & getDjzdx () 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

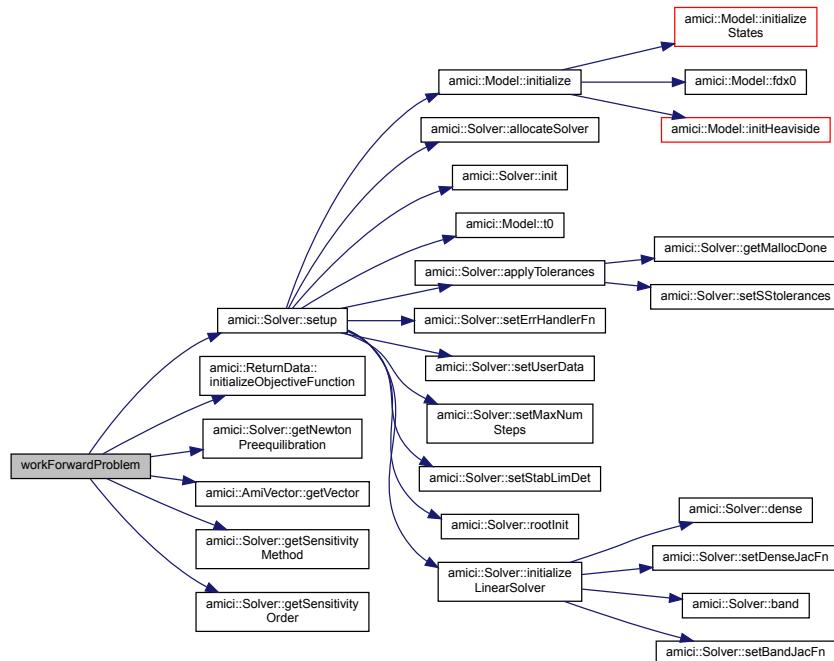
#### 10.7.3.1 workForwardProblem()

```
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()

```
realtype getTime ( ) 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()

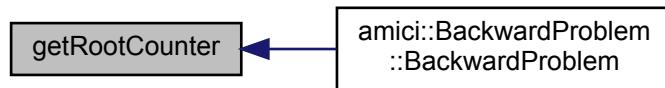
```
int getRootCounter ( ) const  
accessor for iroot
```

#### 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

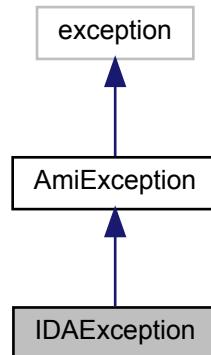
Definition at line 146 of file forwardproblem.h.

## 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 &amp; 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

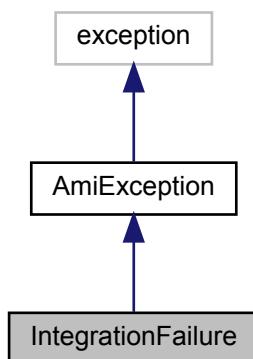
Definition at line 135 of file exception.h.

## 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

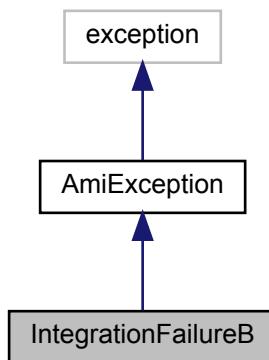
Definition at line 149 of file exception.h.

## 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

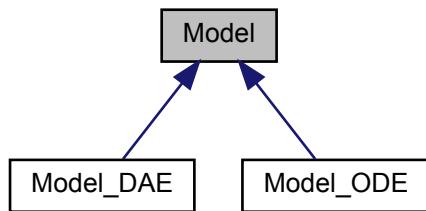
Definition at line 171 of file exception.h.

## 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 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 fxdotdp (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 fdyp (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 fdeltax (const int ie, const realtype t, const AmiVector *x, const AmiVector *xdot, const AmiVector *xdot←_old)`
  - `void fdeltaxs (const int ie, const realtype t, const AmiVector *x, const AmiVectorArray *sx, const AmiVector *xdot, const AmiVector *xdot_old)`
  - `void fdeltaxB (const int ie, const realtype t, const AmiVector *x, const AmiVector *xB, const AmiVector *xdot, const AmiVector *xdot_old)`
  - `void fdeltaqB (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< **realtype** > &dJydx, **ReturnData** \*rdata)
- void **fdJydp** (const int it, const **ExpData** \*edata, **ReturnData** \*rdata)
- void **fdJydx** (std::vector< **realtype** > \*dJydx, const int it, const **ExpData** \*edata, const **ReturnData** \*rdata)
- void **fsJz** (const int nroots, const std::vector< **realtype** > &dJzdx, **AmiVectorArray** \*sx, **ReturnData** \*rdata)
- void **fdJzdp** (const int nroots, **realtype** t, const **ExpData** \*edata, const **ReturnData** \*rdata)
- void **fdJzdx** (std::vector< **realtype** > \*dJzdx, const int nroots, **realtype** 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< **realtype** > const & **getParameters** () const
 

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

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

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

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

*Sets the fixedParameter member.*
- **realtype getFixedParameterById** (std::string const &par\_id) const
 

*Get value of fixed parameter with the specified Id.*
- **realtype getFixedParameterByName** (std::string const &par\_name) const

- Get value of fixed parameter with the specified name, if multiple parameters have the same name, the first parameter with matching name is returned.*
- void `setFixedParameterByld` (std::string const &par\_id, **realtype** value)
 

*Set value of first fixed parameter with the specified id.*
  - int `setFixedParametersByldRegex` (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.*
  - 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 `unscaleParameters` (double \*bufferUnscaled) const
 

*Remove parameter scaling according to the parameter scaling in pscale.*
  - void `requireSensitivitiesForAllParameters` ()
 

*Require computation of sensitivities for all parameters p [0..np[ in natural order.*
  - void `fw` (const **realtype** t, const N\_Vector x)
 

*Recurring terms in xdot.*
  - void `fdwdp` (const **realtype** t, const N\_Vector x)
 

*Recurring terms in xdot, parameter derivative.*
  - void `fdwdx` (const **realtype** t, const N\_Vector 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 fixedParmeters are to be reinitialized after preequilibration and presimulation*
- bool `getReinitializeFixedParameterInitialStates` () const  
*get whether initial states depending on fixedParmeters 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)`
- virtual void `fdydp (realtype *dydp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)`
- virtual void `fdydx (realtype *dydx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)`
- 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 **fdeltax** (**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 **fdeltasx** (**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 **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)
- 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)
- 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** \*wdx, 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

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

#### Friends

- template<class Archive>  
`void boost::serialization::serialize (Archive &ar, 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 nadap,
    const int nnz,
    const int ubw,
    const int lbw,
    amici::SecondOrderMode o2mode,
    const std::vector< amici::realtyp > & p,
    std::vector< amici::realtyp > k,
    const std::vector< int > & plist,
    std::vector< amici::realtyp > idlist,
    std::vector< int > z2event )
```

constructor with model dimensions

#### Parameters

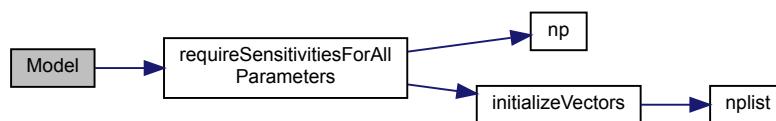
<i>nx</i>	number of state variables
<i>nxtrue</i>	number of state variables of the non-augmented model
<i>ny</i>	number of observables
<i>nytrue</i>	number of observables of the non-augmented model
<i>nz</i>	number of event observables
<i>nztrue</i>	number of event observables of the non-augmented model
<i>ne</i>	number of events
<i>nJ</i>	number of objective functions

**Parameters**

<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 604 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 679 of file model.cpp.

#### 10.11.2.4 ~Model()

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

destructor

Definition at line 737 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\\_ODE](#), and [Model\\_DAE](#).

## 10.11.3.4 froot()

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

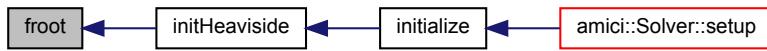
Root function

**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]
```

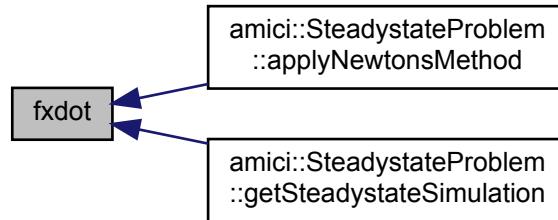
Residual function

**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](#).

Here is the caller graph for this function:



### 10.11.3.6 fJ()

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

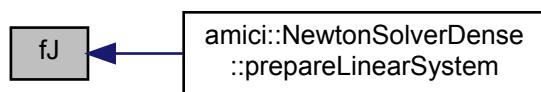
Dense Jacobian function

#### Parameters

<i>t</i>	time
<i>cj</i>	scaling factor (inverse of timestep, DAE only)
<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\\_ODE](#), and [Model\\_DAE](#).

Here is the caller graph for this function:



### 10.11.3.7 fJSparse()

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

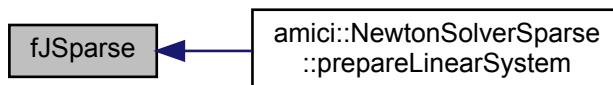
Sparse Jacobian function

#### 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:



### 10.11.3.8 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:

**10.11.3.9 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:



## 10.11.3.10 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>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)

Implemented in [Model\\_ODE](#), and [Model\\_DAE](#).

Here is the caller graph for this function:



## 10.11.3.11 fx0() [1/2]

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

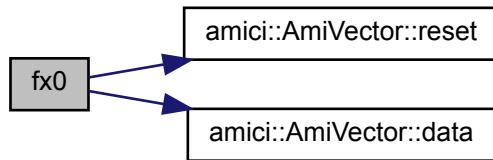
Initial states

## Parameters

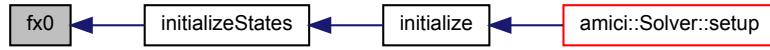
<i>x</i>	pointer to state variables
----------	----------------------------

Definition at line 758 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.12 fx0\_fixedParameters() [1/2]

```
void fx0_fixedParameters (
    AmiVector * x )
```

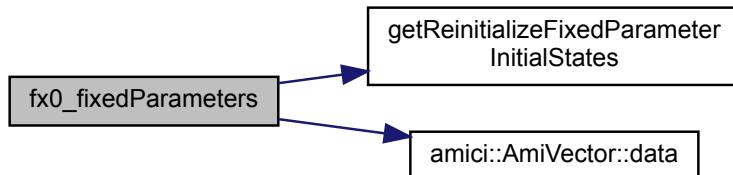
Sets only those initial states that are specified via fixedParmeters

##### Parameters

<code>x</code>	pointer to state variables
----------------	----------------------------

Definition at line 763 of file model.cpp.

Here is the call graph for this function:



## 10.11.3.13 fdx0()

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

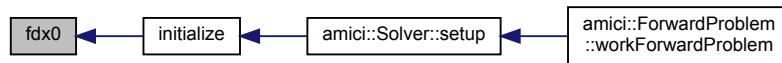
Initial value for time derivative of states (only necessary for DAEs)

**Parameters**

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

Definition at line 776 of file model.cpp.

Here is the caller graph for this function:



## 10.11.3.14 fsx0() [1/2]

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

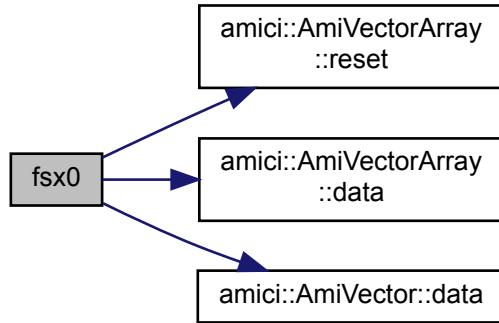
Initial value for initial state sensitivities

**Parameters**

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

Definition at line 778 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.15 fsx0\_fixedParameters() [1/2]

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

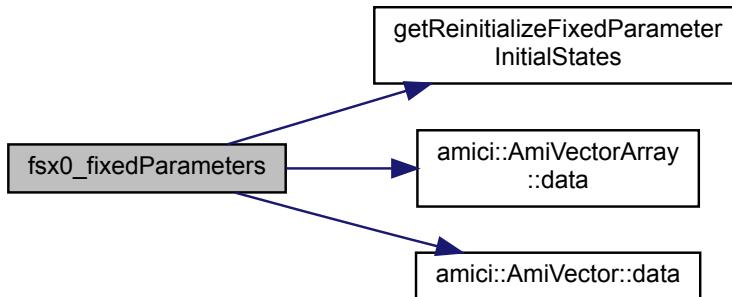
Sets only those initial states sensitivities that are affected from fx0 fixedParameters

##### Parameters

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

Definition at line 768 of file model.cpp.

Here is the call graph for this function:



**10.11.3.16 fsdx0()**

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

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

Definition at line 784 of file model.cpp.

**10.11.3.17 fstau() [1/2]**

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

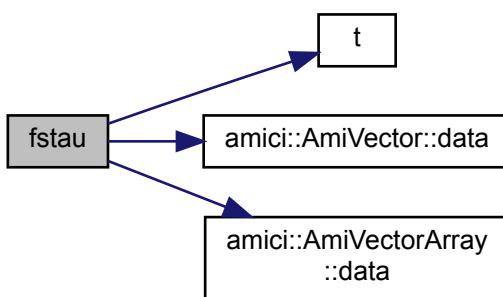
Sensitivity of event timepoint, total derivative

**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 786 of file model.cpp.

Here is the call graph for this function:



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

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

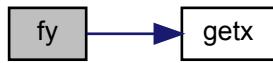
Observables / measurements

**Parameters**

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

Definition at line 793 of file model.cpp.

Here is the call graph for this function:

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

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

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

**Parameters**

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

Definition at line 800 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.20 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 818 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.21 fz() [1/2]

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

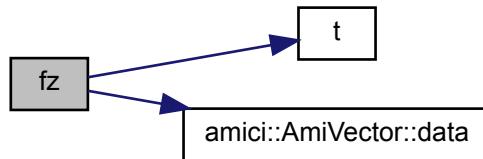
Event-resolved output

**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 826 of file model.cpp.

Here is the call graph for this function:

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

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

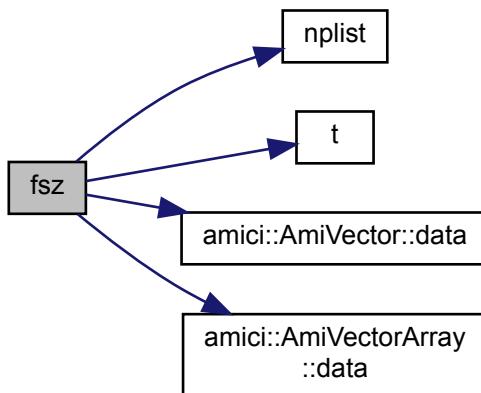
Sensitivity of z, total derivative

**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 830 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.23 frz() [1/2]

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

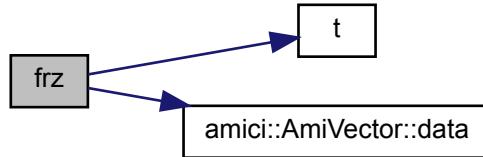
Event root function of events (equal to froot but does not include non-output events)

##### Parameters

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

Definition at line 836 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.24 fsrz() [1/2]

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

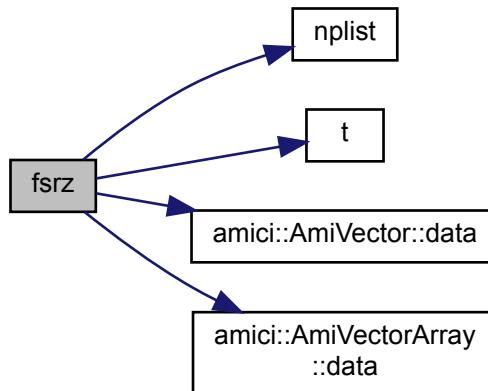
Sensitivity of rz, total derivative

##### 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 840 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.25 fdzdp() [1/2]

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

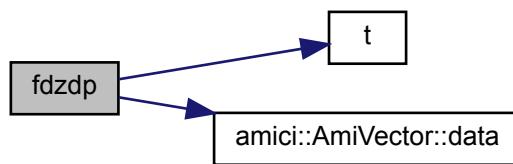
partial derivative of event-resolved output z w.r.t. to model parameters p

##### Parameters

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

Definition at line 846 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.26 fdzdx() [1/2]

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

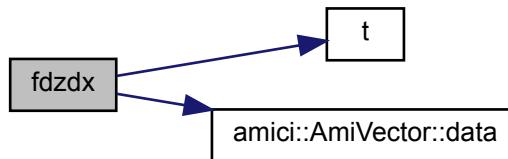
partial derivative of event-resolved output z w.r.t. to model states x

#### Parameters

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

Definition at line 853 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.27 fdrzdp() [1/2]

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

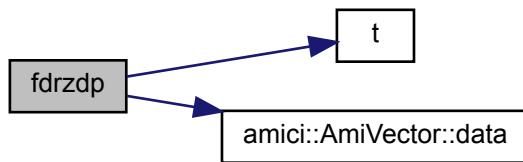
Sensitivity of event-resolved root output w.r.t. to model parameters p

#### Parameters

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

Definition at line 858 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.28 fdrzdx() [1/2]

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

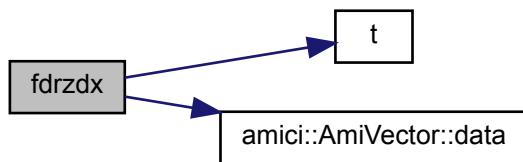
Sensitivity of event-resolved measurements rz w.r.t. to model states x

##### Parameters

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

Definition at line 866 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.29 fdeltax() [1/2]

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

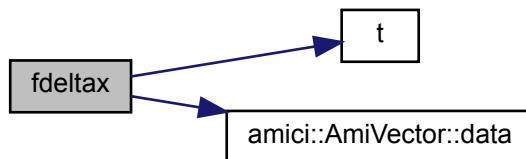
State update functions for events

#### 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 871 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.30 fdeltasx() [1/2]

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

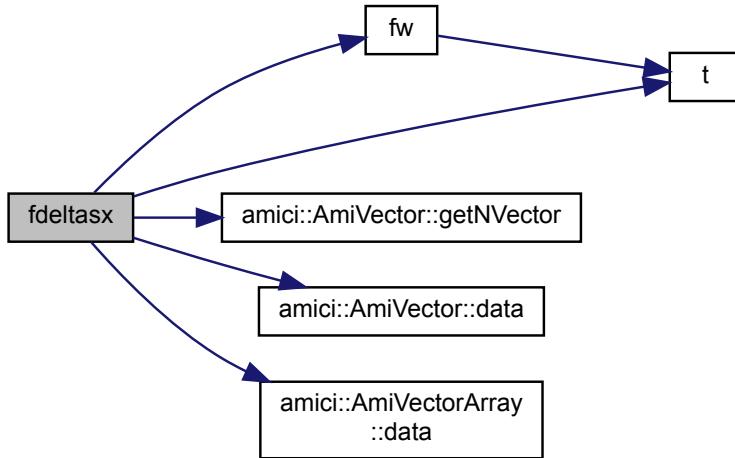
Sensitivity update functions for events, total derivative

#### 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 877 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.31 fdeltaxB() [1/2]

```

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

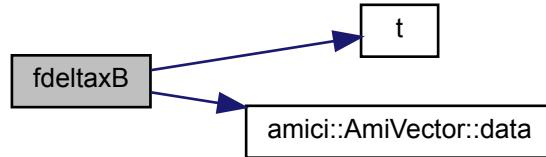
Adjoint state update functions for events

##### Parameters

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

Definition at line 886 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.32 fdeltaqB() [1/2]

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

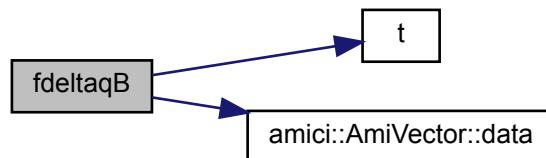
Quadrature state update functions for events

##### 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 892 of file model.cpp.

Here is the call graph for this function:



10.11.3.33 **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 900 of file model.cpp.

Here is the call graph for this function:

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

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

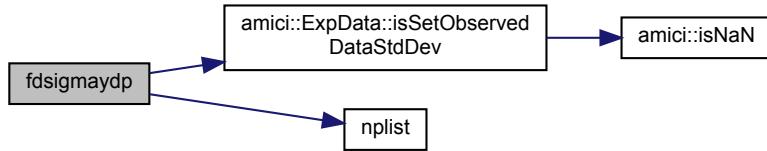
partial derivative of standard deviation of measurements w.r.t. model

#### 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 919 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.35 `fsigmaz()` [1/2]

```

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

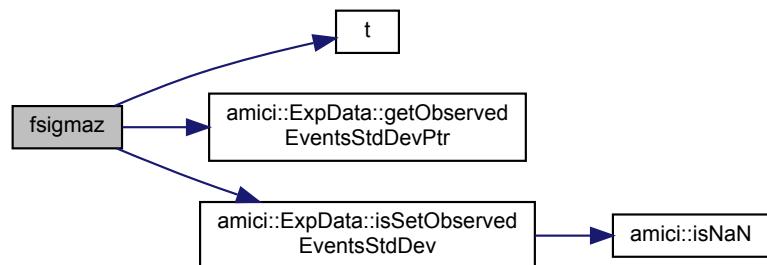
Standard deviation of events

##### 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 948 of file model.cpp.

Here is the call graph for this function:



10.11.3.36 `fdsigmazdp()` [1/2]

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

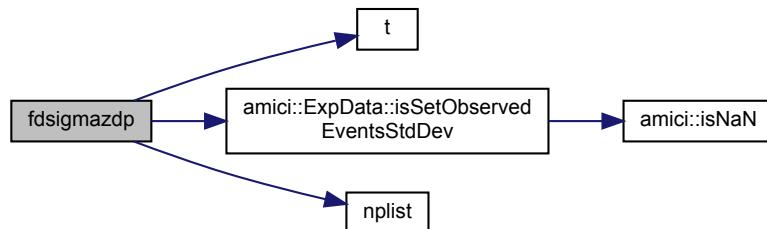
Sensitivity of standard deviation of events measurements w.r.t. model parameters p

**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 966 of file model.cpp.

Here is the call graph for this function:

10.11.3.37 `fJy()` [1/2]

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

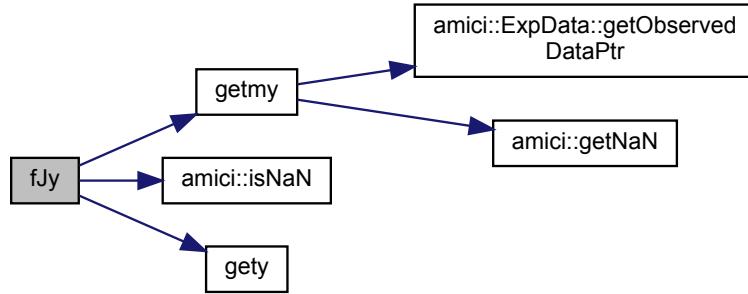
negative log-likelihood of measurements y

**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 991 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.38 fJz() [1/2]

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

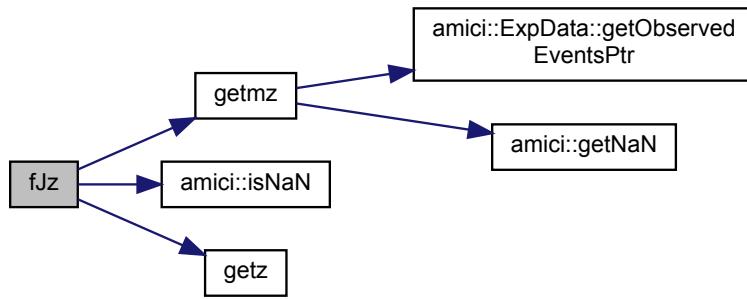
negative log-likelihood of event-resolved measurements z

##### 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 1003 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.39 fJrz() [1/2]

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

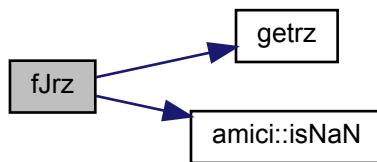
regularization of negative log-likelihood with roots of event-resolved measurements rz

##### 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 1015 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.40 fdJydy() [1/2]

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

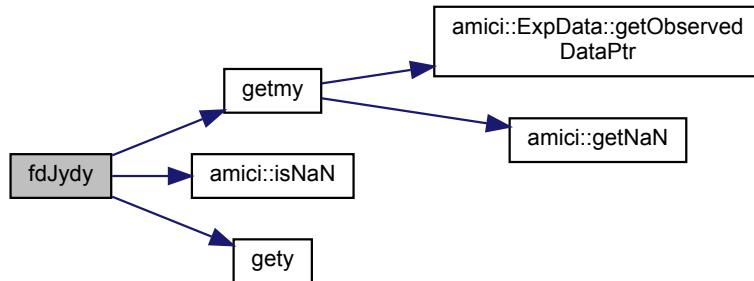
partial derivative of time-resolved measurement negative log-likelihood Jy

#### 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 1027 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.41 fdJydsigma() [1/2]

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

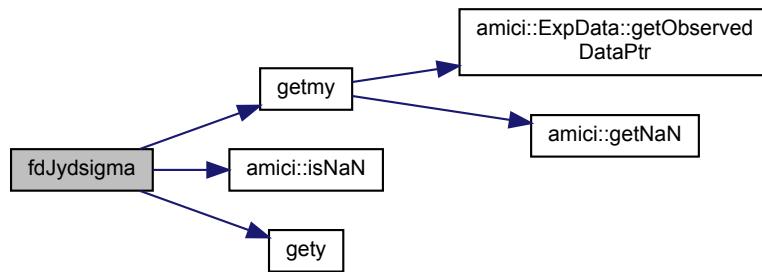
Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigma

#### 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 1047 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.42 fdJzdz() [1/2]

```

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

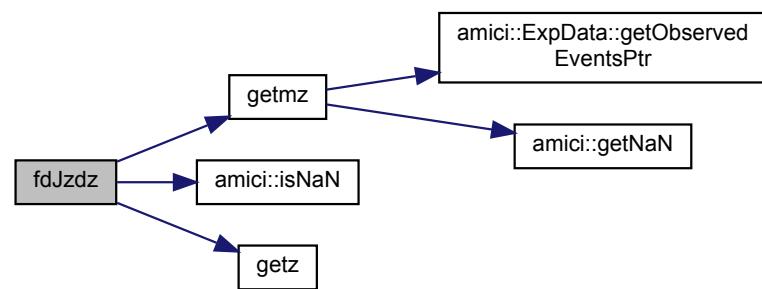
partial derivative of event measurement negative log-likelihood Jz

##### 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 1067 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.43 fdJzdsigma() [1/2]

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

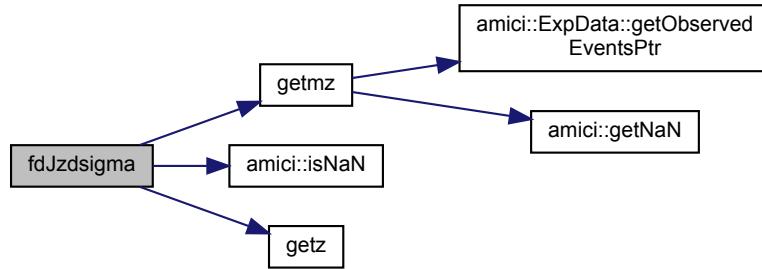
Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz

#### 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 1078 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.44 fdJrzdz() [1/2]

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

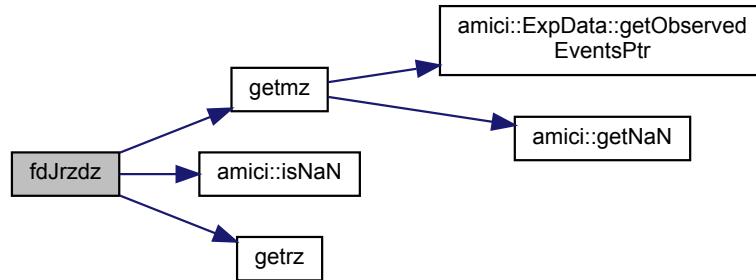
partial derivative of event measurement negative log-likelihood Jz

#### 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 1089 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.45 fdJrzdsigma() [1/2]

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

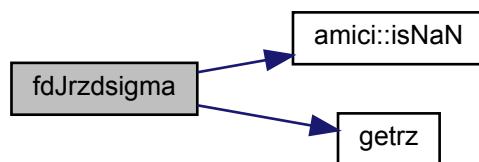
Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz

##### 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 1100 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.46 fsy()

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

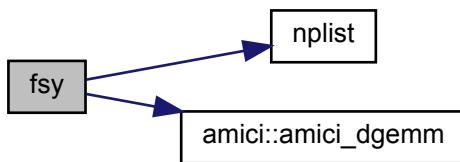
Sensitivity of measurements y, total derivative  $sy = dydx * sx + dydp$

#### Parameters

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

Definition at line 13 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.47 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 29 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.48 fsJy()

```

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

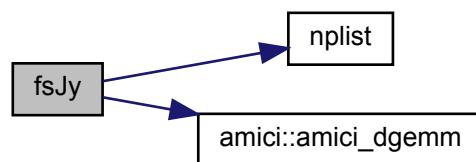
Sensitivity of time-resolved measurement negative log-likelihood Jy, total derivative

##### 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 37 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.49 fdJydp()

```

void fdJydp (
    const int it,
  
```

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

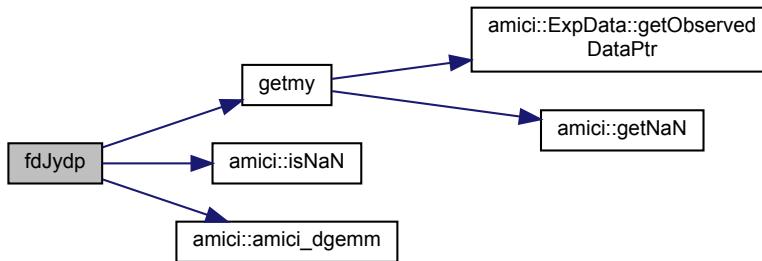
Compute sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. parameters for the given time-point. Add result to respective fields in rdata.

#### 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 66 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.50 fdJydx()

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

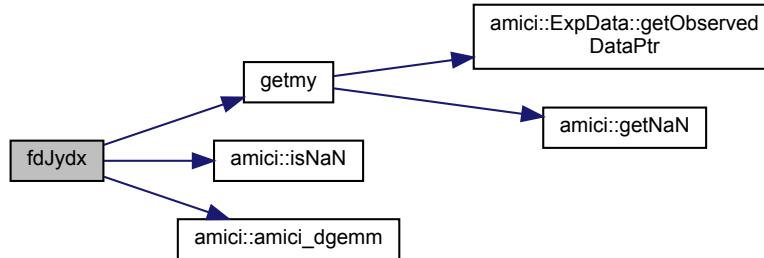
Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. state variables

#### 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 114 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.51 fsJz()

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

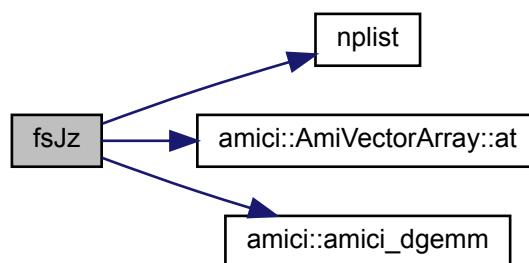
Sensitivity of event-resolved measurement negative log-likelihood Jz, total derivative

##### 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 133 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.52 fdJzdp()

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

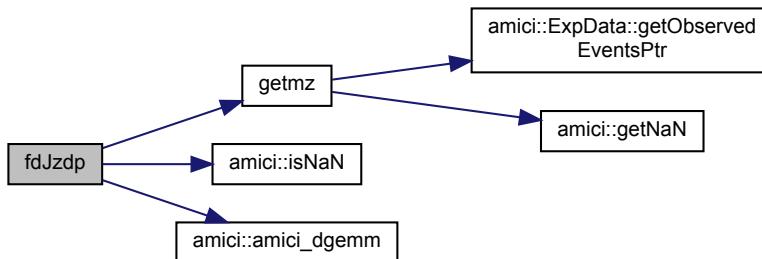
Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. parameters

#### 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 167 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.53 fdJzdx()

```
void fdJzdx (
    std::vector< realtype > * dJzdx,
    const int nroots,
    realtype t,
    const ExpData * edata,
    const ReturnData * rdata )
```

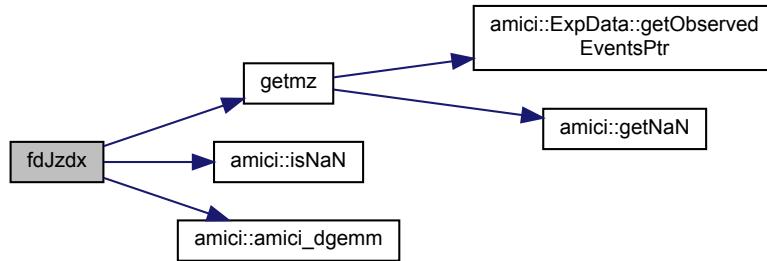
Sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. state variables

#### 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 203 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.54 initialize()

```
void initialize (
    AmiVector * x,
    AmiVector * dx )
```

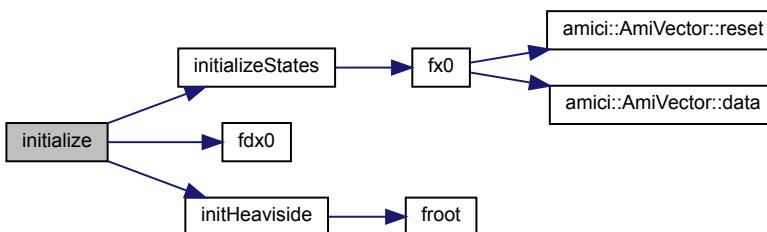
initialization of model properties

##### Parameters

<i>x</i>	pointer to state variables
<i>dx</i>	pointer to time derivative of states (DAE only)

Definition at line 226 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.55 initializeStates()

```
void initializeStates (
    AmiVector * x )
```

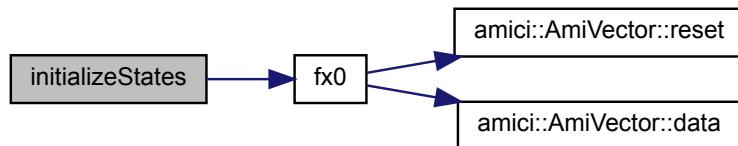
initialization of initial states

##### Parameters

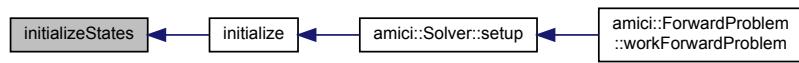
x	pointer to state variables
---	----------------------------

Definition at line 237 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.11.3.56 initHeaviside()

```
void initHeaviside (
    AmiVector * x,
    AmiVector * dx )
```

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

## Parameters

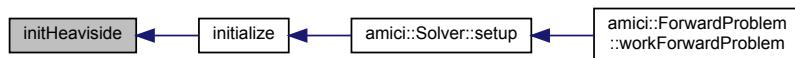
<i>x</i>	pointer to state variables
<i>dx</i>	pointer to time derivative of states (DAE only)

Definition at line 248 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.11.3.57 nplist()

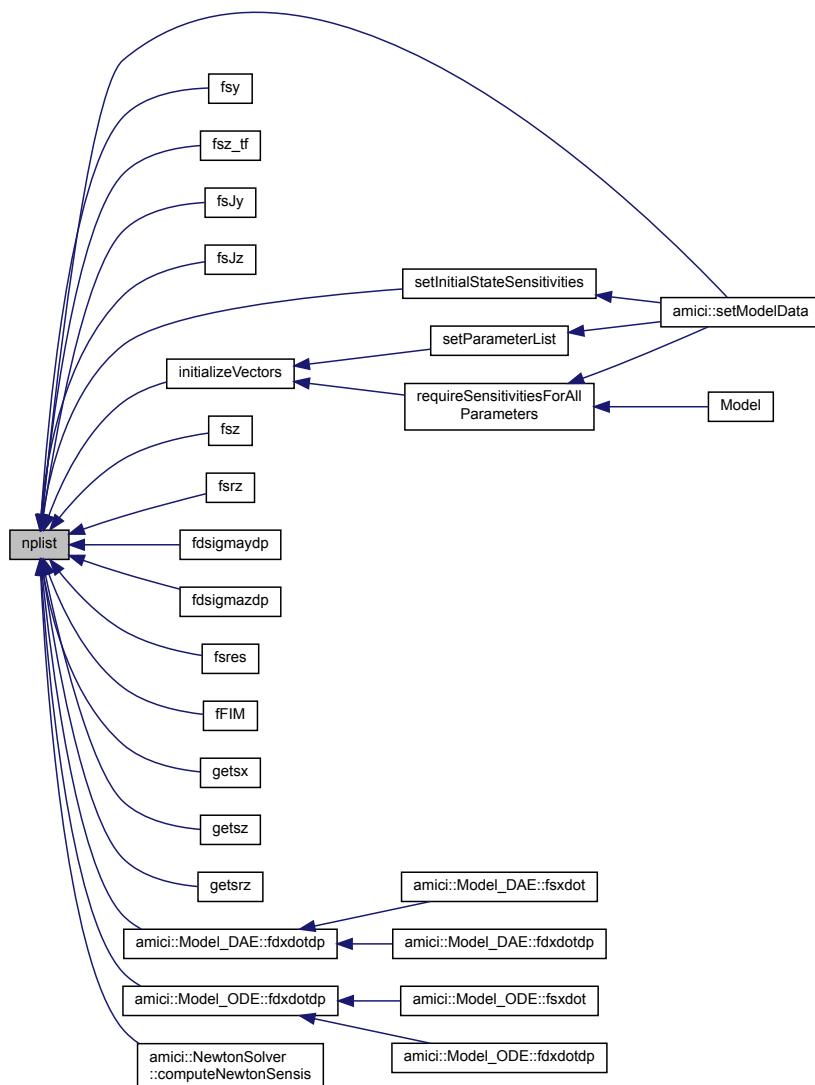
```
int nplist ( ) const
```

## Returns

length of sensitivity index vector

Definition at line 265 of file model.cpp.

Here is the caller graph for this function:



### 10.11.3.58 np()

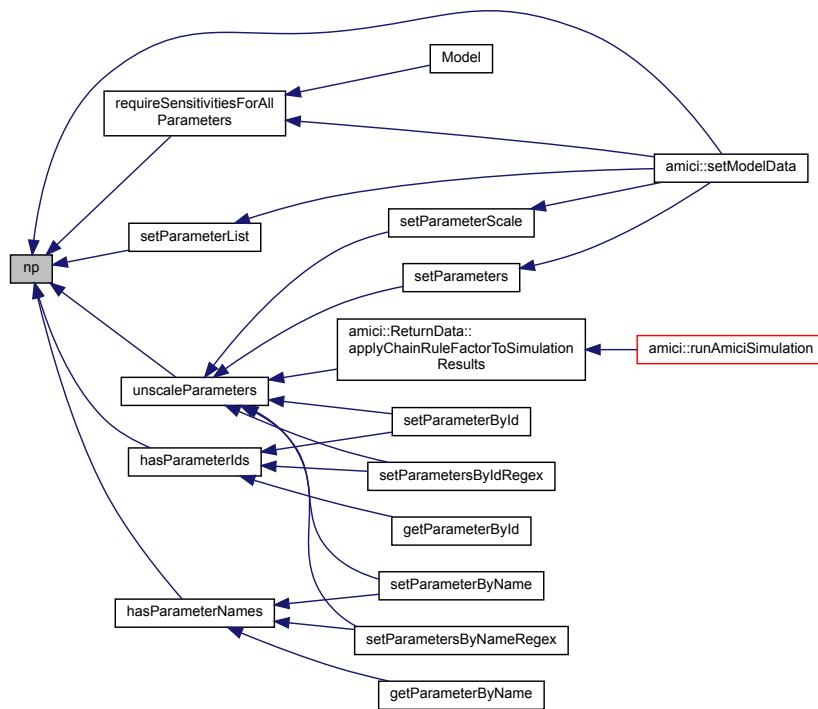
```
int np( ) const
```

#### Returns

length of parameter vector

Definition at line 269 of file model.cpp.

Here is the caller graph for this function:



### 10.11.3.59 nk()

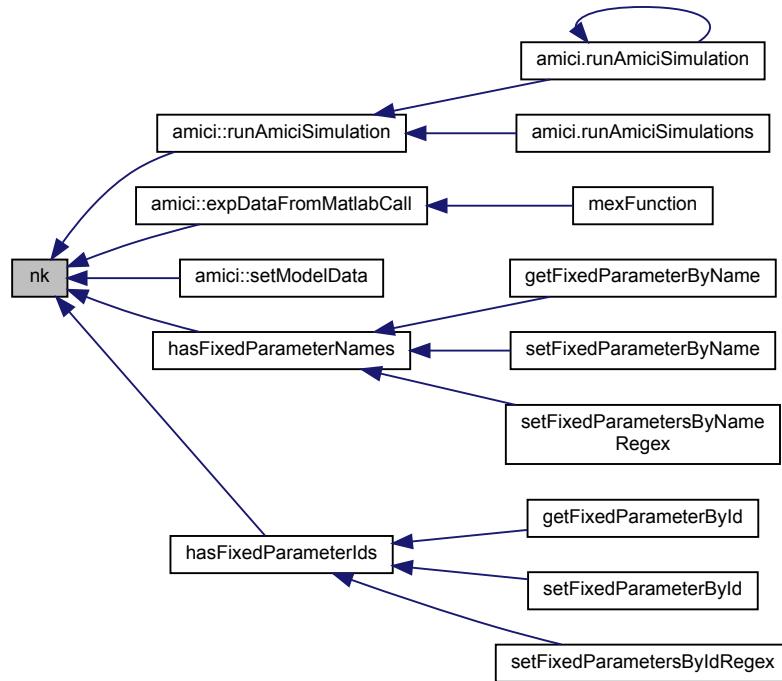
```
int nk ( ) const
```

**Returns**

length of constant vector

Definition at line 273 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.60 k()**

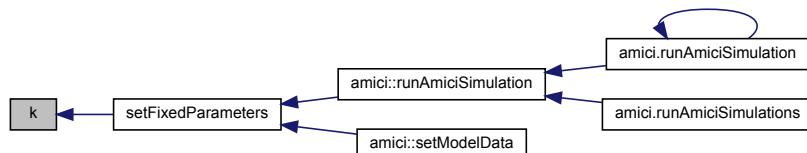
```
const double * k( ) const
```

**Returns**

pointer to constants array

Definition at line 277 of file model.cpp.

Here is the caller graph for this function:



**10.11.3.61 nMaxEvent()**

```
int nMaxEvent ( ) const
```

**Returns**

maximum number of events that may occur for each type

Definition at line 281 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.62 setNMaxEvent()**

```
void setNMaxEvent ( int nmaxevent )
```

**Parameters**

<i>nmaxevent</i>	maximum number of events that may occur for each type
------------------	---

Definition at line 285 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.63 nt()**

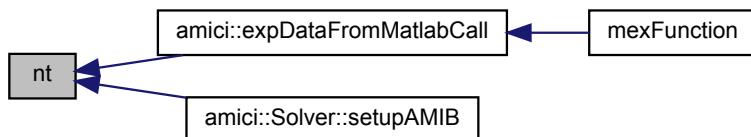
```
int nt ( ) const
```

**Returns**

number of timepoints

Definition at line 289 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.64 getParameterScale()**

```
const std::vector< ParameterScaling > & getParameterScale ( ) const
```

**Returns**

vector of parameter scale

Definition at line 293 of file model.cpp.

**10.11.3.65 setParameterScale() [1/2]**

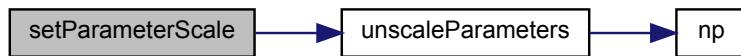
```
void setParameterScale (
    ParameterScaling pscale )
```

**Parameters**

<i>pscale</i>	scalar parameter scale for all parameters
---------------	---

Definition at line 297 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.66 setParameterScale() [2/2]

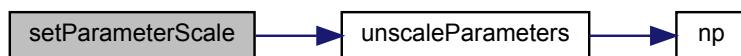
```
void setParameterScale (const std::vector< ParameterScaling > & pscale )
```

##### Parameters

<i>pscale</i>	vector of parameter scales
---------------	----------------------------

Definition at line 303 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.67 getParameters()

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

**Returns**

The user-set parameters (see also `getUnscaledParameters`)

Definition at line 309 of file `model.cpp`.

**10.11.3.68 setParameters()**

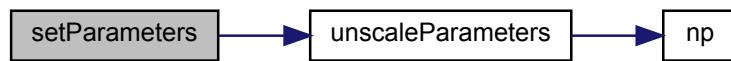
```
void setParameters (
    std::vector< realltype > const & p )
```

**Parameters**

<i>p</i>	vector of parameters
----------	----------------------

Definition at line 392 of file `model.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.69 getUnscaledParameters()**

```
const std::vector< realltype > & getUnscaledParameters ( ) const
```

**Returns**

unscaled parameters

Definition at line 454 of file `model.cpp`.

**10.11.3.70 getFixedParameters()**

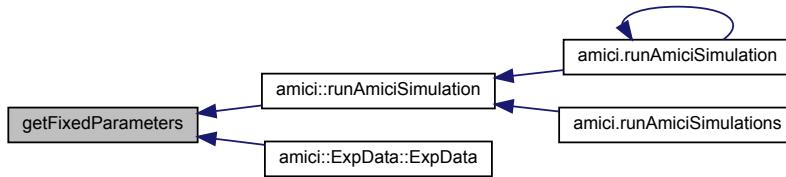
```
const std::vector< realtypes > & getFixedParameters ( ) const
```

**Returns**

vector of fixed parameters

Definition at line 458 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.71 setFixedParameters()**

```
void setFixedParameters ( realtypes > const & k )
```

**Parameters**

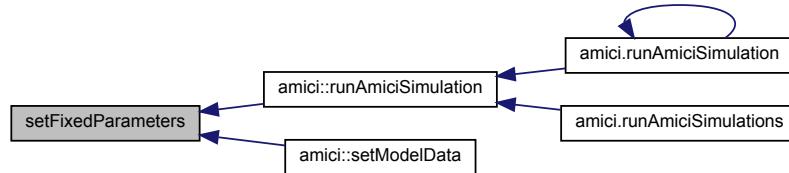
<i>k</i>	vector of fixed parameters
----------	----------------------------

Definition at line 484 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.72 getFixedParameterById()

```
realtype getFixedParameterById (
    std::string const & par_id ) const
```

##### Parameters

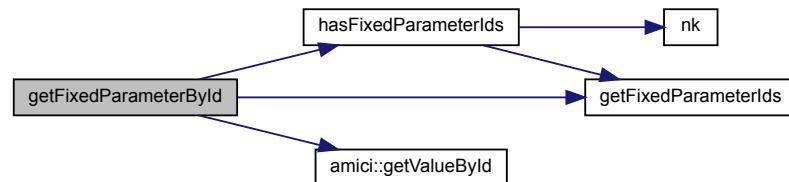
<i>par_id</i>	parameter id
---------------	--------------

##### Returns

parameter value

Definition at line 462 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.73 getFixedParameterByName()

```
realtype getFixedParameterByName (
    std::string const & par_name ) const
```

**Parameters**

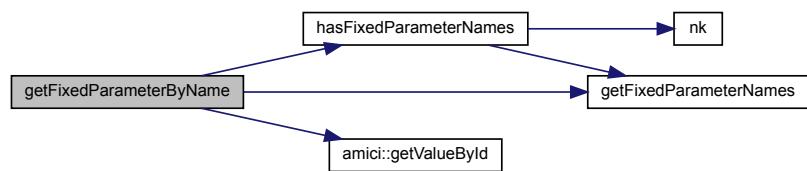
<i>par_name</i>	parameter name
-----------------	----------------

**Returns**

parameter value

Definition at line 473 of file model.cpp.

Here is the call graph for this function:

**10.11.3.74 setFixedParameterById()**

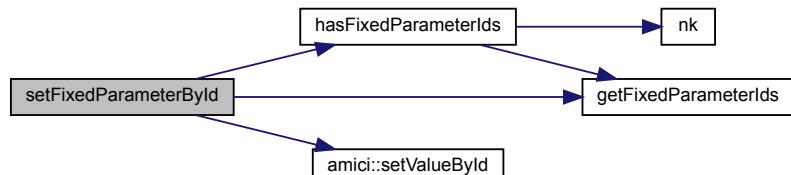
```
void setFixedParameterById (
    std::string const & par_id,
    realltype value )
```

**Parameters**

<i>par_id</i>	fixed parameter id
<i>value</i>	fixed parameter value

Definition at line 490 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.75 setFixedParametersByIdRegex()

```
int setFixedParametersByIdRegex (
    std::string const & par_id_regex,
    realtype value )
```

#### Parameters

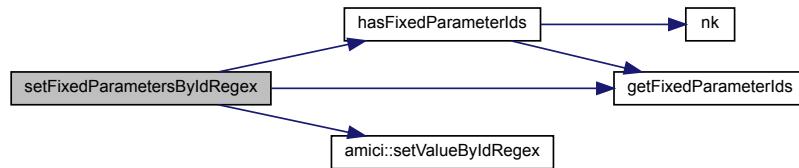
<i>par_id_regex</i>	fixed parameter name regex
<i>value</i>	fixed parameter value

#### Returns

number of fixed parameter ids that matched the regex

Definition at line 502 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.76 setFixedParameterByName()

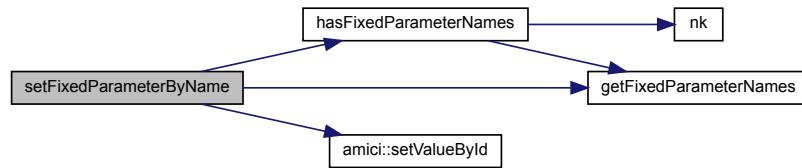
```
void setFixedParameterByName (
    std::string const & par_name,
    realtype value )
```

#### Parameters

<i>par_name</i>	fixed parameter id
<i>value</i>	fixed parameter value

Definition at line 514 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.77 setFixedParametersByNameRegex()

```
int setFixedParametersByNameRegex (
    std::string const & par_name_regex,
    realtype value )
```

##### Parameters

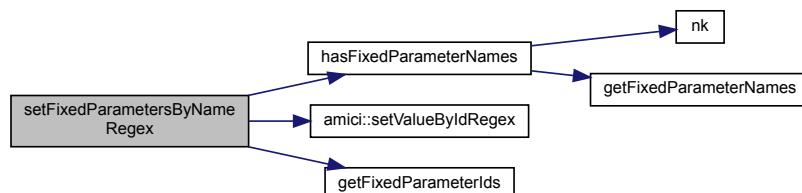
<code>par_name_regex</code>	fixed parameter name regex
<code>value</code>	fixed parameter value

##### Returns

number of fixed parameter names that matched the regex

Definition at line 526 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.78 getTimepoints()

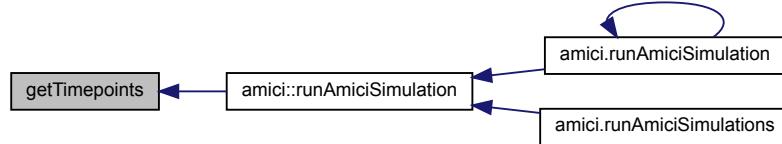
```
std::vector< realtype > const & getTimepoints( ) const
```

**Returns**

```
timepoint vector
```

Definition at line 538 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.79 setTimpoints()**

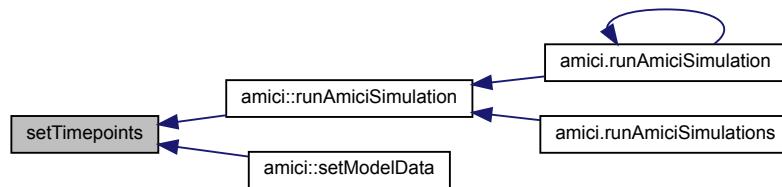
```
void setTimepoints (
    std::vector< realltype > const & ts )
```

**Parameters**

<i>ts</i>	timepoint vector
-----------	------------------

Definition at line 542 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.80 t()**

```
double t (
    int idx ) const
```

## Parameters

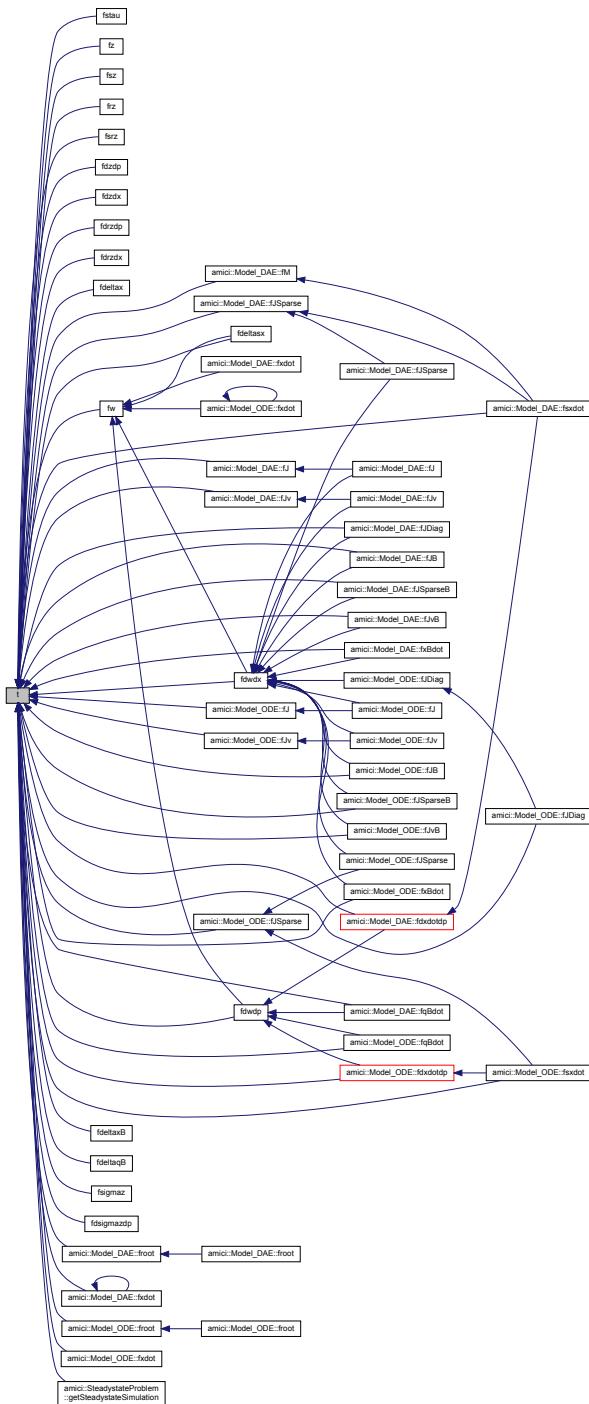
*idx* timepoint index

## Returns

timepoint

Definition at line 548 of file model.cpp.

Here is the caller graph for this function:



### 10.11.3.81 getParameterList()

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

#### Returns

list of parameter indices

Definition at line 552 of file model.cpp.

### 10.11.3.82 setParameterList()

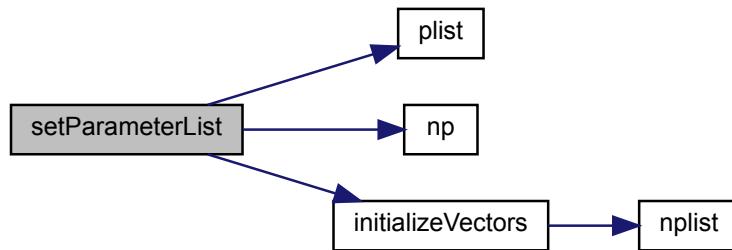
```
void setParameterList (   
    std::vector< int > const & plist )
```

#### Parameters

<i>plist</i>	list of parameter indices
--------------	---------------------------

Definition at line 556 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



**10.11.3.83 getInitialStates()**

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

**Returns**

initial state vector

Definition at line 565 of file model.cpp.

**10.11.3.84 setInitialStates()**

```
void setInitialStates ( realtype > const & x0 )
```

**Parameters**

<i>x0</i>	initial state vector
-----------	----------------------

Definition at line 569 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.85 getInitialStateSensitivities()**

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

**Returns**

vector of initial state sensitivities

Definition at line 578 of file model.cpp.

**10.11.3.86 setInitialStateSensitivities()**

```
void setInitialStateSensitivities ( realtype > const & sx0 )
```

**Parameters**

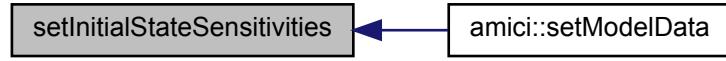
<code>sx0</code>	vector of initial state sensitivities
------------------	---------------------------------------

Definition at line 582 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.87 t0()**

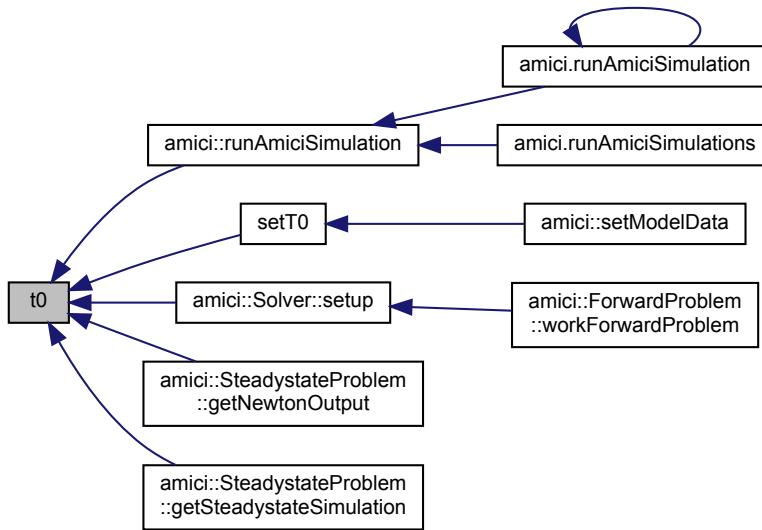
```
double t0 () const
```

**Returns**

simulation start time

Definition at line 591 of file model.cpp.

Here is the caller graph for this function:



#### 10.11.3.88 setT0()

```
void setT0 ( double t0 )
```

##### Parameters

<code>t0</code>	simulation start time
-----------------	-----------------------

Definition at line 595 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.89 plist()

```
int plist (
    int pos ) const
```

##### Parameters

<i>pos</i>	<i>index</i>
------------	--------------

##### Returns

entry

Definition at line 599 of file model.cpp.

Here is the caller graph for this function:



#### 10.11.3.90 unscaleParameters()

```
void unscaleParameters (
    double * bufferUnscaled ) const
```

##### Parameters

<i>out</i>	<i>bufferUnscaled</i>	unscaled parameters are written to the array
		<b>Type:</b> double

**Returns**

status flag indicating success of execution

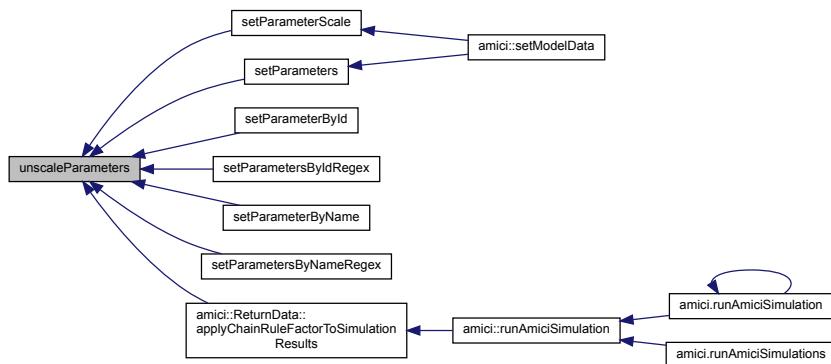
**Type:** int

Definition at line 1259 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.91 fw() [1/2]**

```

void fw (
    const realltype t,
    const N_Vecor x )
  
```

**Parameters**

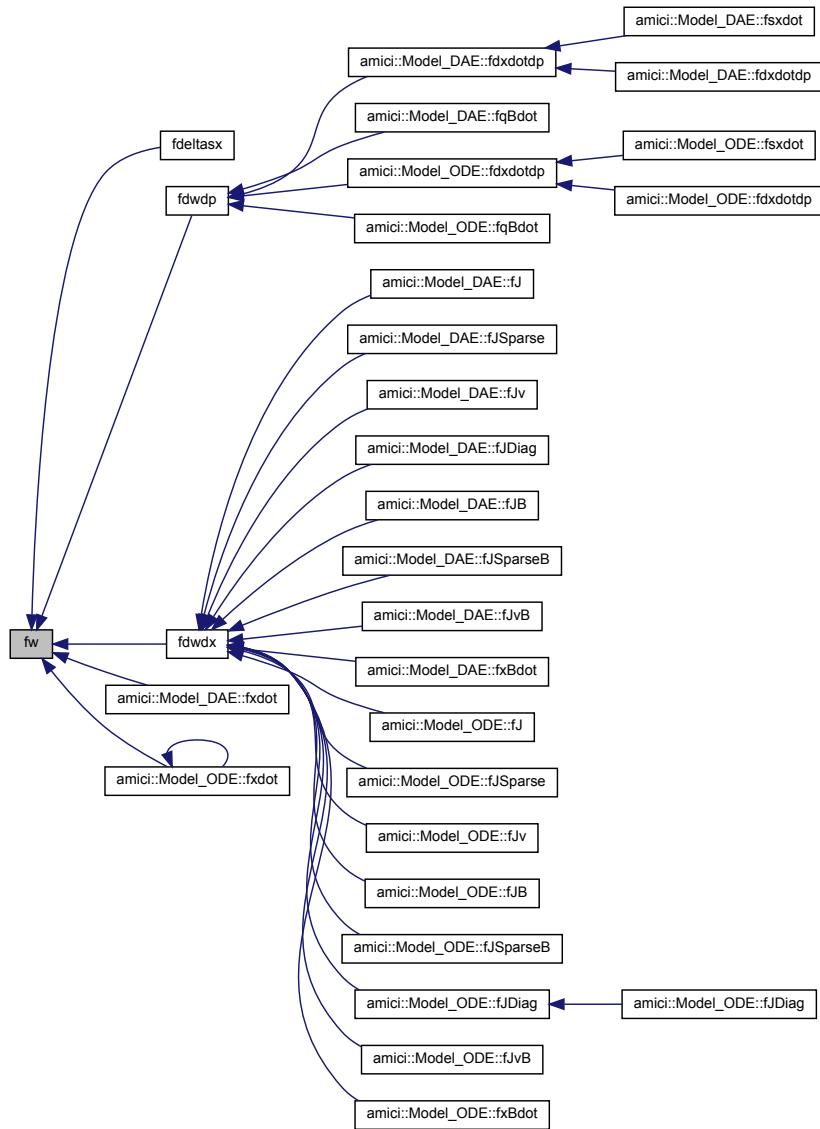
<i>t</i>	timepoint
<i>x</i>	Vector with the states

Definition at line 1110 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.11.3.92 fdwdp() [1/2]

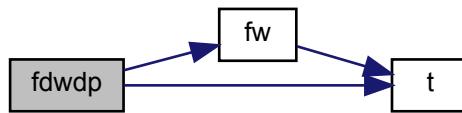
```
void fdwdp (
    const realltype t,
    const N_Vector x )
```

## Parameters

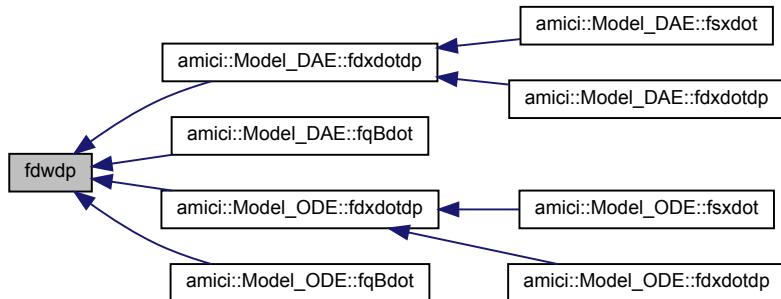
<i>t</i>	timepoint
<i>x</i>	Vector with the states

Definition at line 1115 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.11.3.93 fdwdx() [1/2]

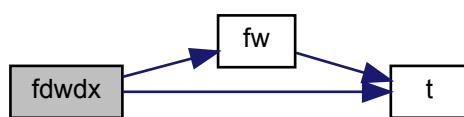
```
void fdwdx (
    const realltype t,
    const N_Vector x )
```

**Parameters**

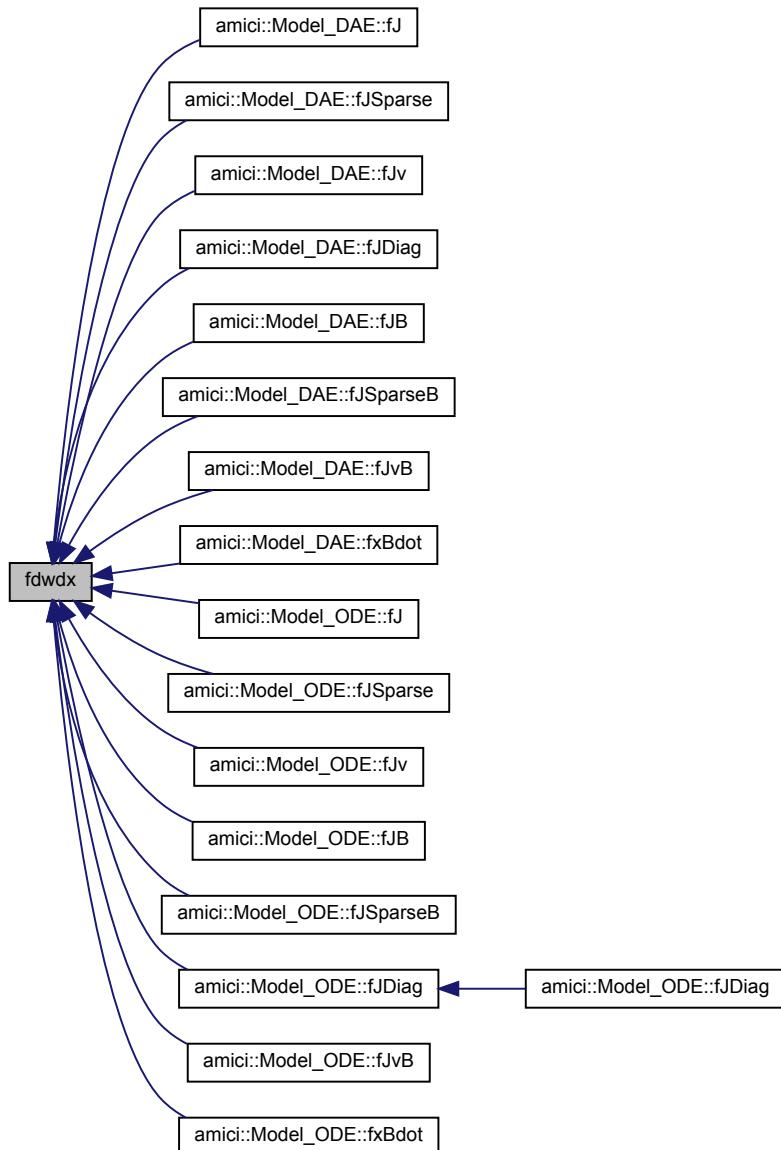
<i>t</i>	timepoint
<i>x</i>	Vector with the states

Definition at line 1121 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.94 fres()

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

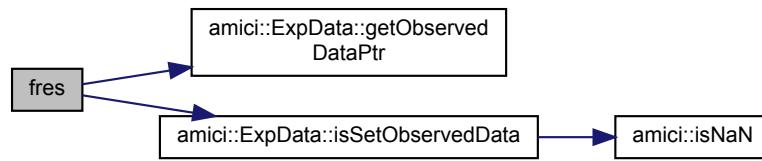
residual function

**Parameters**

<i>it</i>	time index
<i>rdata</i>	<a href="#">ReturnData</a> instance to which result will be written
<i>edata</i>	<a href="#">ExpData</a> instance containing observable data

Definition at line 1127 of file model.cpp.

Here is the call graph for this function:

**10.11.3.95 fchi2()**

```
void fchi2 (
    const int it,
    ReturnData * rdata )
```

chi-squared function

**Parameters**

<i>it</i>	time index
<i>rdata</i>	<a href="#">ReturnData</a> instance to which result will be written

Definition at line 1142 of file model.cpp.

**10.11.3.96 fsres()**

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

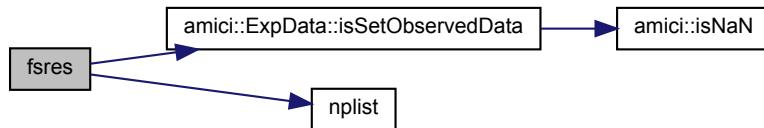
residual sensitivity function

**Parameters**

<i>it</i>	time index
<i>rdata</i>	<a href="#">ReturnData</a> instance to which result will be written
<i>edata</i>	<a href="#">ExpData</a> instance containing observable data

Definition at line 1152 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.97 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 1168 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.98 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 1184 of file model.cpp.

**10.11.3.99 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 1190 of file model.cpp.

**10.11.3.100 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 1217 of file model.cpp.

**10.11.3.101 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**

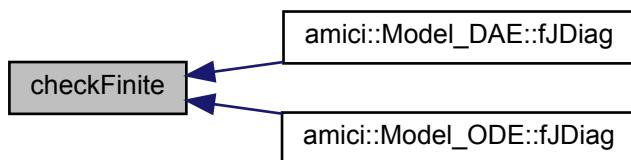
AMICI\_RECOVERABLE\_ERROR if a NaN/Inf value was found, AMICI\_SUCCESS otherwise

Definition at line 1245 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.11.3.102 hasParameterNames()

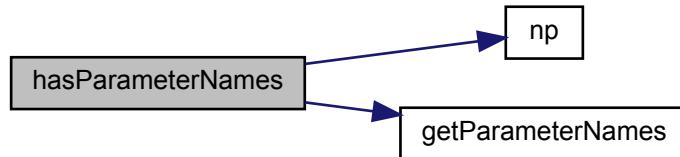
```
virtual bool hasParameterNames( ) const [virtual]
```

**Returns**

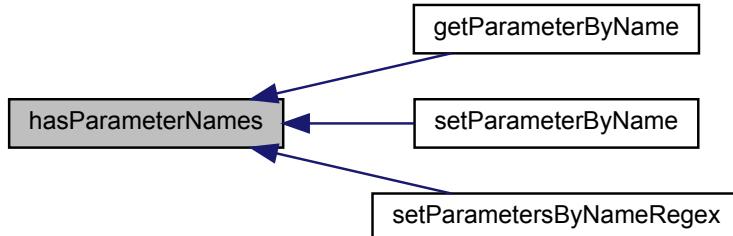
boolean indicating whether parameter names were set

Definition at line 866 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.103 getParameterNames()**

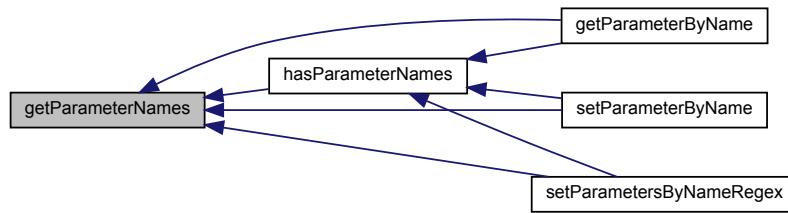
```
virtual std::vector<std::string> getParameterNames ( ) const [virtual]
```

**Returns**

the names

Definition at line 872 of file model.h.

Here is the caller graph for this function:



#### 10.11.3.104 hasStateNames()

```
virtual bool hasStateNames( ) const [virtual]
```

##### Returns

boolean indicating whether state names were set

Definition at line 878 of file model.h.

Here is the call graph for this function:



#### 10.11.3.105 getStateNames()

```
virtual std::vector<std::string> getStateNames( ) const [virtual]
```

**Returns**

the names

Definition at line 884 of file model.h.

Here is the caller graph for this function:

**10.11.3.106 hasFixedParameterNames()**

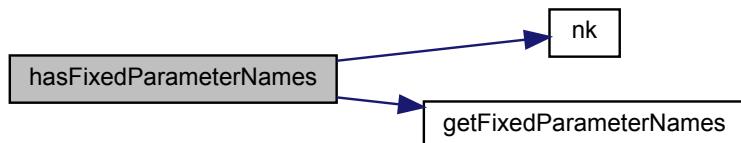
```
virtual bool hasFixedParameterNames () const [virtual]
```

**Returns**

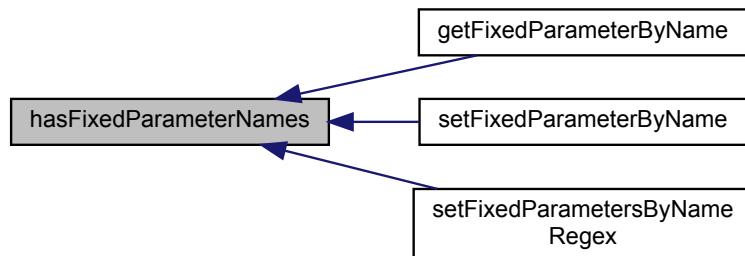
boolean indicating whether fixed parameter names were set

Definition at line 890 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:



**10.11.3.107 getFixedParameterNames()**

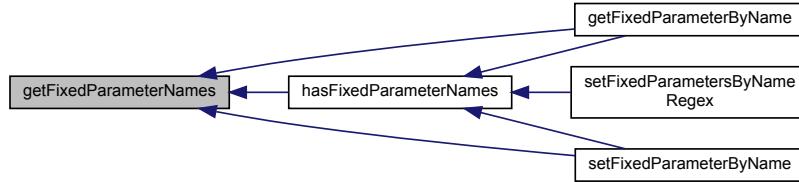
```
virtual std::vector<std::string> getFixedParameterNames( ) const [virtual]
```

**Returns**

the names

Definition at line 896 of file model.h.

Here is the caller graph for this function:

**10.11.3.108 hasObservableNames()**

```
virtual bool hasObservableNames( ) const [virtual]
```

**Returns**

boolean indicating whether observable names were set

Definition at line 902 of file model.h.

Here is the call graph for this function:



### 10.11.3.109 getObservableNames()

```
virtual std::vector<std::string> getObservableNames ( ) const [virtual]
```

#### Returns

the names

Definition at line 908 of file model.h.

Here is the caller graph for this function:



### 10.11.3.110 hasParameterIds()

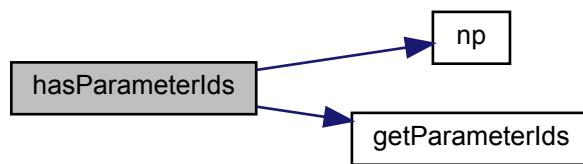
```
virtual bool hasParameterIds ( ) const [virtual]
```

#### Returns

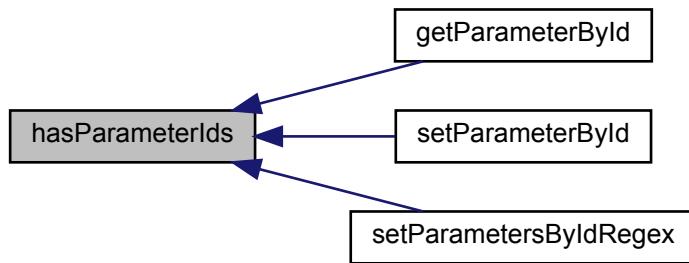
boolean indicating whether parameter ids were set

Definition at line 914 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.111 `getParameterIds()`

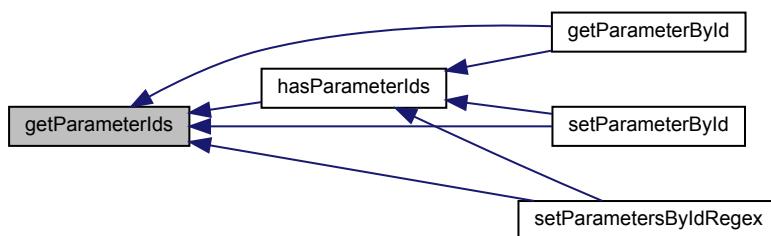
```
virtual std::vector<std::string> getParameterIds () const [virtual]
```

##### Returns

the ids

Definition at line 920 of file `model.h`.

Here is the caller graph for this function:



#### 10.11.3.112 `getParameterById()`

```
realtype getParameterById (
    std::string const & par_id ) const
```

**Parameters**

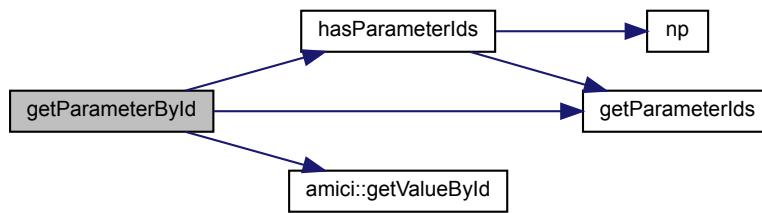
$par_{\leftarrow}$ $_id$	parameter id
-----------------------------	--------------

**Returns**

parameter value

Definition at line 376 of file model.cpp.

Here is the call graph for this function:

**10.11.3.113 getParameterByName()**

```
realtype getParameterByName (  
    std::string const & par_name ) const
```

**Parameters**

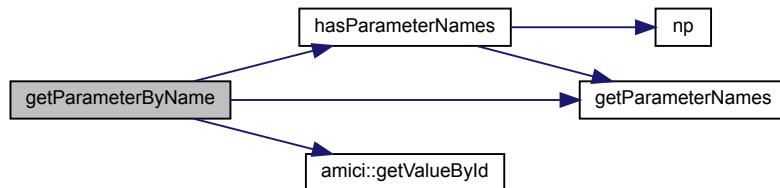
$par\_name$	parameter name
-------------	----------------

**Returns**

parameter value

Definition at line 382 of file model.cpp.

Here is the call graph for this function:

**10.11.3.114 setParameterById()**

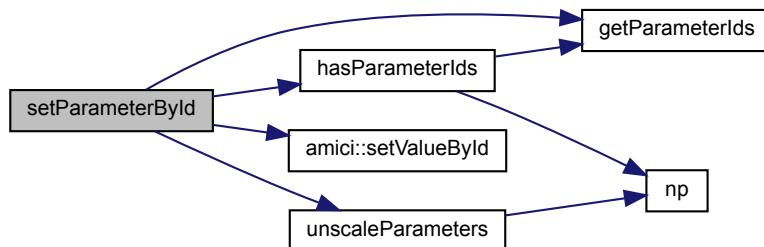
```
void setParameterById (
    std::string const & par_id,
    realtype value )
```

**Parameters**

<i>par_id</i>	parameter id
<i>value</i>	parameter value

Definition at line 400 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.115 setParametersByIdRegex()

```
int setParametersByIdRegex (
    std::string const & par_id_regex,
    realtype value )
```

#### Parameters

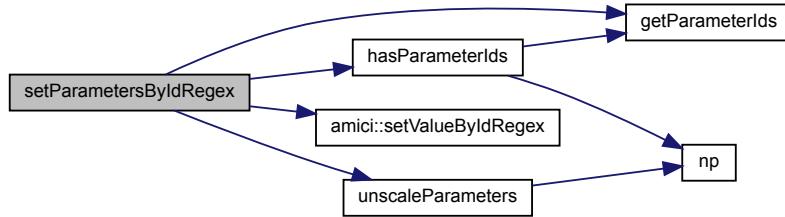
<i>par_id_regex</i>	parameter id regex
<i>value</i>	parameter value

#### Returns

number of parameter ids that matched the regex

Definition at line 413 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.116 setParameterByName()

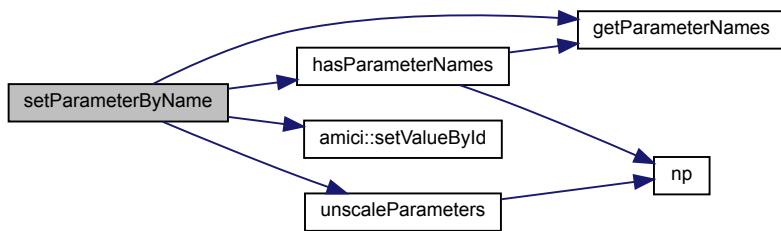
```
void setParameterByName (
    std::string const & par_name,
    realtype value )
```

#### Parameters

<i>par_name</i>	parameter name
<i>value</i>	parameter value

Definition at line 426 of file model.cpp.

Here is the call graph for this function:



#### 10.11.3.117 setParametersByNameRegex()

```
int setParametersByNameRegex (
    std::string const & par_name_regex,
    realltype value )
```

##### Parameters

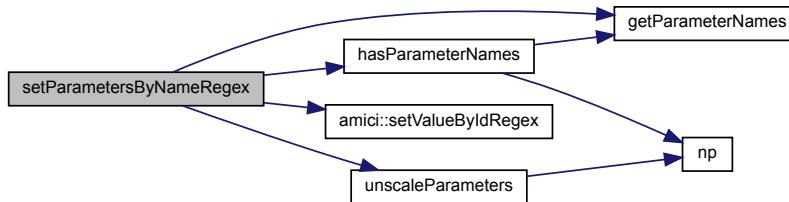
<code>par_name_regex</code>	parameter name regex
<code>value</code>	parameter value

##### Returns

number of fixed parameter names that matched the regex

Definition at line 439 of file `model.cpp`.

Here is the call graph for this function:



### 10.11.3.118 hasStateIds()

```
virtual bool hasStateIds ( ) const [virtual]
```

#### Returns

Definition at line 972 of file model.h.

Here is the call graph for this function:



### 10.11.3.119 getStateIds()

```
virtual std::vector<std::string> getStateIds ( ) const [virtual]
```

#### Returns

the ids

Definition at line 978 of file model.h.

Here is the caller graph for this function:



**10.11.3.120 hasFixedParameterIds()**

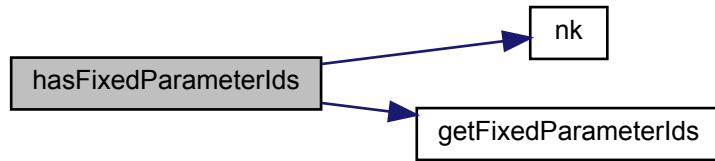
```
virtual bool hasFixedParameterIds () const [virtual]
```

**Returns**

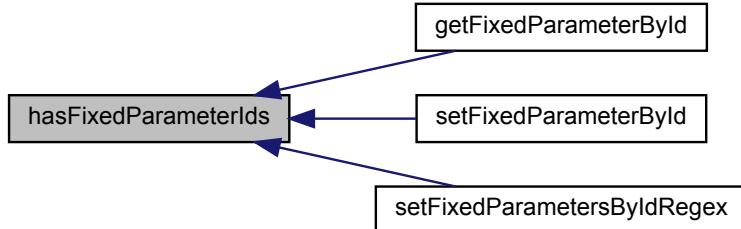
boolean indicating whether fixed parameter ids were set

Definition at line 986 of file model.h.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.11.3.121 getFixedParameterIds()**

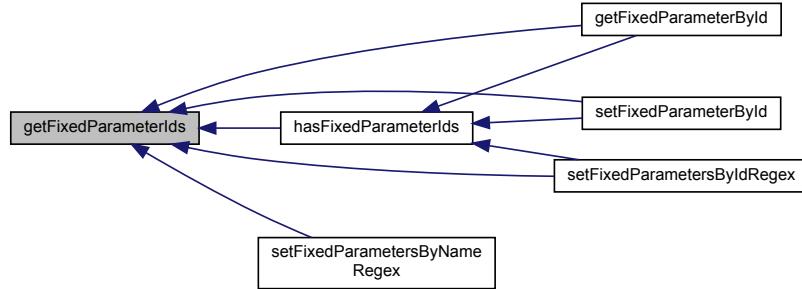
```
virtual std::vector<std::string> getFixedParameterIds () const [virtual]
```

**Returns**

the ids

Definition at line 992 of file model.h.

Here is the caller graph for this function:

**10.11.3.122 hasObservableIds()**

```
virtual bool hasObservableIds ( ) const [virtual]
```

**Returns**

boolean indicating whether observale ids were set

Definition at line 1000 of file model.h.

Here is the call graph for this function:



**10.11.3.123 getObservableIds()**

```
virtual std::vector<std::string> getObservableIds() const [virtual]
```

**Returns**

the ids

Definition at line 1006 of file model.h.

Here is the caller graph for this function:

**10.11.3.124 setSteadyStateSensitivityMode()**

```
void setSteadyStateSensitivityMode( const SteadyStateSensitivityMode mode )
```

**Parameters**

<i>mode</i>	steadyStateSensitivityMode
-------------	----------------------------

Definition at line 1014 of file model.h.

**10.11.3.125 getSteadyStateSensitivityMode()**

```
SteadyStateSensitivityMode getSteadyStateSensitivityMode() const
```

**Returns**

flag value

Definition at line 1022 of file model.h.

**10.11.3.126 setReinitializeFixedParameterInitialStates()**

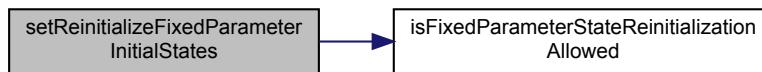
```
void setReinitializeFixedParameterInitialStates( bool flag )
```

**Parameters**

<i>flag</i>	true/false
-------------	------------

Definition at line 1031 of file model.h.

Here is the call graph for this function:

**10.11.3.127 getReinitializeFixedParameterInitialStates()**

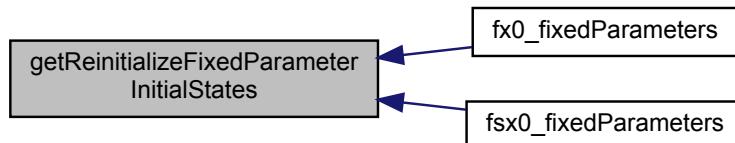
```
bool getReinitializeFixedParameterInitialStates ( ) const
```

**Returns**

*flag* true/false

Definition at line 1045 of file model.h.

Here is the caller graph for this function:

**10.11.3.128 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 1125 of file model.h.

**10.11.3.129 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 1135 of file model.h.

**10.11.3.130 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 1146 of file model.h.

#### 10.11.3.131 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 1157 of file model.h.

#### 10.11.3.132 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 1172 of file model.h.

### 10.11.3.133 fy() [2/2]

```
virtual void fy (
    realtype * y,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [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

Definition at line 1184 of file model.h.

### 10.11.3.134 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 ) [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

Definition at line 1197 of file model.h.

### 10.11.3.135 fdydx() [2/2]

```
virtual void fdydx (
    realtype * dydx,
    const realtype t,
    const realtype * x,
    const realtype * p,
    const realtype * k,
    const realtype * h ) [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

Definition at line 1209 of file model.h.

### 10.11.3.136 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 1222 of file model.h.

10.11.3.137 **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 1237 of file model.h.

10.11.3.138 **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 1250 of file model.h.

#### 10.11.3.139 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 1265 of file model.h.

#### 10.11.3.140 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 1279 of file model.h.

**10.11.3.141 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 1292 of file model.h.

**10.11.3.142 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 1306 of file model.h.

**10.11.3.143 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 1319 of file model.h.

**10.11.3.144 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 1334 of file model.h.

**10.11.3.145 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 1354 of file model.h.

## 10.11.3.146 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 1372 of file model.h.

## 10.11.3.147 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 1390 of file model.h.

**10.11.3.148 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 1401 of file model.h.

**10.11.3.149 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 1412 of file model.h.

#### 10.11.3.150 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 1422 of file model.h.

#### 10.11.3.151 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 1433 of file model.h.

#### 10.11.3.152 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 1446 of file model.h.

#### 10.11.3.153 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 1458 of file model.h.

#### 10.11.3.154 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 1470 of file model.h.

#### 10.11.3.155 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 1483 of file model.h.

#### 10.11.3.156 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 1498 of file model.h.

#### 10.11.3.157 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 1512 of file model.h.

#### 10.11.3.158 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 1527 of file model.h.

#### 10.11.3.159 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 1540 of file model.h.

#### 10.11.3.160 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 1554 of file model.h.

#### 10.11.3.161 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 1567 of file model.h.

#### 10.11.3.162 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 1579 of file model.h.

#### 10.11.3.163 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 1591 of file model.h.

#### 10.11.3.164 getmy()

```
void getmy (
    const int it,
    const ExpData * edata) [protected]
```

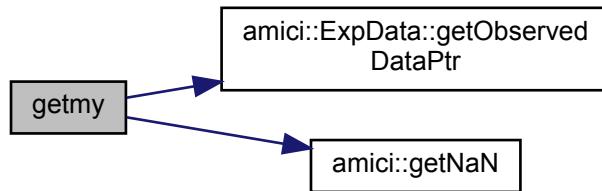
create my slice at timepoint

**Parameters**

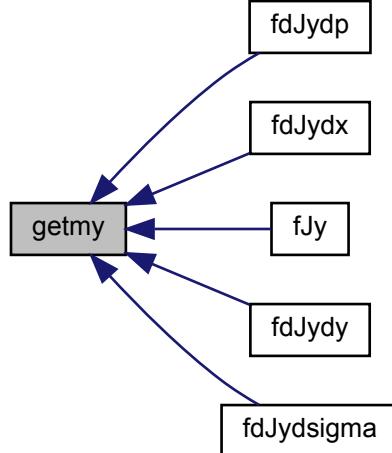
<i>it</i>	timepoint index
<i>edata</i>	pointer to experimental data instance

Definition at line 1197 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.11.3.165 `getmz()`

```
void getmz (
    const int nroots,
    const ExpData * edata ) [protected]
```

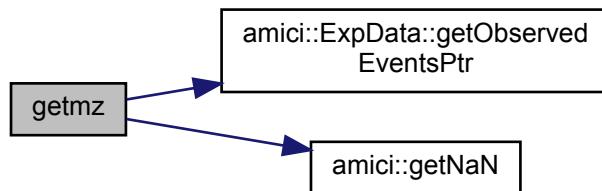
create mz slice at event

**Parameters**

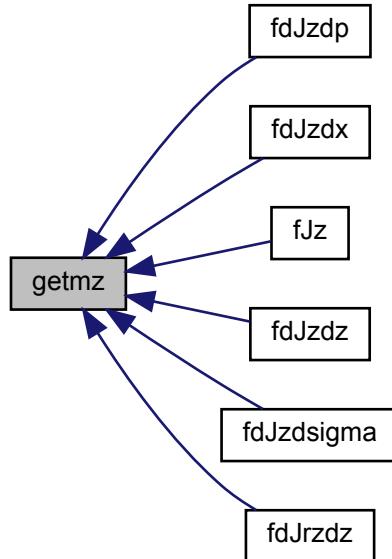
<i>nroots</i>	event occurrence
<i>edata</i>	pointer to experimental data instance

Definition at line 1221 of file model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.11.3.166 gety()

```
const realtype * gety (
    const int it,
    const ReturnData * rdata ) const [protected]
```

create y slice at timepoint

#### Parameters

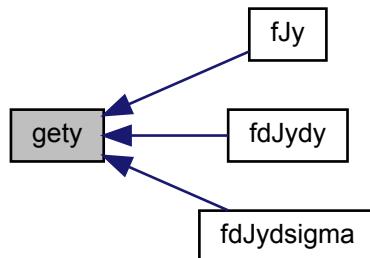
<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

#### Returns

y y-slice from rdata instance

Definition at line 1213 of file model.cpp.

Here is the caller graph for this function:



### 10.11.3.167 getx()

```
const realtype * getx (
    const int it,
    const ReturnData * rdata ) const [protected]
```

create x slice at timepoint

#### Parameters

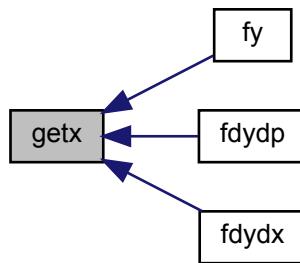
<i>it</i>	timepoint index
<i>rdata</i>	pointer to return data instance

**Returns**

x x-slice from rdata instance

Definition at line 1205 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.168 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 1209 of file model.cpp.

Here is the call graph for this function:



### 10.11.3.169 getz()

```
const realtype * getz (
    const int nroots,
    const ReturnData * rdata ) const [protected]
```

create z slice at event

#### Parameters

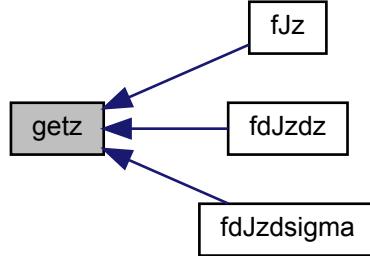
<i>nroots</i>	event occurrence
<i>rdata</i>	pointer to return data instance

#### Returns

z slice

Definition at line 1229 of file model.cpp.

Here is the caller graph for this function:



### 10.11.3.170 getrz()

```
const realtype * getrz (
    const int nroots,
    const ReturnData * rdata ) const [protected]
```

create rz slice at event

#### Parameters

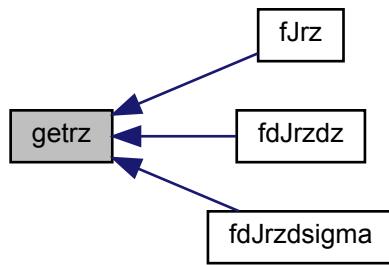
<i>nroots</i>	event occurrence
<i>rdata</i>	pointer to return data instance

**Returns**

rz slice

Definition at line 1233 of file model.cpp.

Here is the caller graph for this function:

**10.11.3.171 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 1237 of file model.cpp.

Here is the call graph for this function:

**10.11.3.172 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 1241 of file model.cpp.

Here is the call graph for this function:



## 10.11.3.173 isFixedParameterStateReinitializationAllowed()

```
virtual bool isFixedParameterStateReinitializationAllowed() const [protected], [virtual]
```

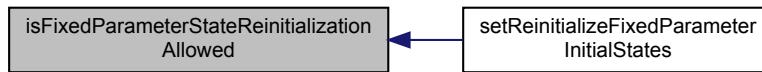
function indicating whether reinitialization of states depending on fixed parameters is permissible

**Returns**

flag indicating whether reinitialization of states depending on fixed parameters is permissible

Definition at line 1662 of file model.h.

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 1282 of file model.cpp.

**10.11.5 Member Data Documentation****10.11.5.1 nx**

const int nx

number of states

Definition at line 1050 of file model.h.

**10.11.5.2 nxtrue**

const int nxtrue

number of states in the unaugmented system

Definition at line 1052 of file model.h.

**10.11.5.3 ny**

const int ny

number of observables

Definition at line 1054 of file model.h.

**10.11.5.4 nytrue**

const int nytrue

number of observables in the unaugmented system

Definition at line 1056 of file model.h.

**10.11.5.5 nz**

```
const int nz
```

number of event outputs

Definition at line 1058 of file model.h.

**10.11.5.6 nztrue**

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 1060 of file model.h.

**10.11.5.7 ne**

```
const int ne
```

number of events

Definition at line 1062 of file model.h.

**10.11.5.8 nw**

```
const int nw
```

number of common expressions

Definition at line 1064 of file model.h.

**10.11.5.9 ndwdx**

```
const int ndwdx
```

number of derivatives of common expressions wrt x

Definition at line 1066 of file model.h.

**10.11.5.10 ndwdp**

const int ndwdp

number of derivatives of common expressions wrt p

Definition at line 1068 of file model.h.

**10.11.5.11 nnz**

const int nnz

number of nonzero entries in jacobian

Definition at line 1070 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 1072 of file model.h.

**10.11.5.13 ubw**

const int ubw

upper bandwith of the jacobian

Definition at line 1074 of file model.h.

**10.11.5.14 lbw**

const int lbw

lower bandwith of the jacobian

Definition at line 1076 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 1079 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 1081 of file model.h.

**10.11.5.17 idlist**

```
const std::vector<realtyp> idlist
```

flag array for DAE equations

Definition at line 1083 of file model.h.

**10.11.5.18 sigmay**

```
std::vector<realtyp> sigmay
```

data standard deviation for current timepoint (dimension: ny)

Definition at line 1086 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 1088 of file model.h.

**10.11.5.20 sigmaz**

```
std::vector<realtyp> sigmaz
```

event standard deviation for current timepoint (dimension: nz)

Definition at line 1090 of file model.h.

**10.11.5.21 dsigmazdp**

```
std::vector<realtyp> dsigmazdp
```

parameter derivative of event standard deviation for current timepoint (dimension: plist x nz, row-major)

Definition at line 1092 of file model.h.

**10.11.5.22 dJydp**

```
std::vector<realtyp> dJydp
```

parameter derivative of data likelihood for current timepoint (dimension: plist x nJ, row-major)

Definition at line 1094 of file model.h.

**10.11.5.23 dJzdp**

```
std::vector<realtyp> dJzdp
```

parameter derivative of event likelihood for current timepoint (dimension: plist x nJ, row-major)

Definition at line 1096 of file model.h.

**10.11.5.24 deltax**

```
std::vector<realtyp> deltax
```

change in x at current timepoint (dimension: nx)

Definition at line 1099 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 1101 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 1103 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 1105 of file model.h.

**10.11.5.28 dxdotdp**

```
std::vector<realtyp> dxdotdp
```

tempory storage of dxdotdp data across functions (dimension: nplist x nx, row-major)

Definition at line 1108 of file model.h.

**10.11.5.29 J**

```
SlsMat J = nullptr [protected]
```

Sparse Jacobian (dimension: nnz)

Definition at line 1668 of file model.h.

**10.11.5.30 my**

```
std::vector<realtype> my [protected]
```

current observable (dimension: nytrue)

Definition at line 1671 of file model.h.

**10.11.5.31 mz**

```
std::vector<realtype> mz [protected]
```

current event measurement (dimension: nztrue)

Definition at line 1673 of file model.h.

**10.11.5.32 dJydy**

```
std::vector<realtype> dJydy [protected]
```

observable derivative of data likelihood (dimension nJ x nytrue x ny, ordering = ?)

Definition at line 1675 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 1677 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 1680 of file model.h.

**10.11.5.35 dJzdsigma**

```
std::vector<realtyp> dJzdsigma [protected]
```

event sigma derivative of event likelihood (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1682 of file model.h.

**10.11.5.36 dJrzdz**

```
std::vector<realtyp> dJrzdz [protected]
```

event ouput derivative of event likelihood at final timepoint (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1684 of file model.h.

**10.11.5.37 dJrzdsigma**

```
std::vector<realtyp> dJrzdsigma [protected]
```

event sigma derivative of event likelihood at final timepoint (dimension nJ x nztrue x nz, ordering = ?)

Definition at line 1686 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 1688 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 1690 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 1692 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 1694 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 1696 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 1699 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 1701 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 1703 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 1705 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 1707 of file model.h.

**10.11.5.48 stau**

```
std::vector<realtyp> stau [protected]
```

tempory storage of stau data across functions (dimension: nplist)

Definition at line 1709 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 1713 of file model.h.

**10.11.5.50 unscaledParameters**

```
std::vector<realtype> unscaledParameters [protected]
```

unscaled parameters (dimension: np)

Definition at line 1716 of file model.h.

**10.11.5.51 originalParameters**

```
std::vector<realtype> originalParameters [protected]
```

original user-provided, possibly scaled parameter array (size np)

Definition at line 1719 of file model.h.

**10.11.5.52 fixedParameters**

```
std::vector<realtype> fixedParameters [protected]
```

constants (dimension: nk)

Definition at line 1722 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 1725 of file model.h.

**10.11.5.54 x0data**

```
std::vector<double> x0data [protected]
```

state initialisation (size nx)

Definition at line 1728 of file model.h.

**10.11.5.55 sx0data**

```
std::vector<realtyp> sx0data [protected]
```

sensitivity initialisation (size nx \* nplist, ordering = ?)

Definition at line 1731 of file model.h.

**10.11.5.56 ts**

```
std::vector<realtyp> ts [protected]
```

timepoints (size nt)

Definition at line 1734 of file model.h.

**10.11.5.57 nmaxevent**

```
int nmaxevent = 10 [protected]
```

maximal number of events to track

Definition at line 1737 of file model.h.

**10.11.5.58 pscale**

```
std::vector<ParameterScaling> pscale [protected]
```

parameter transformation of originalParameters (dimension np)

Definition at line 1740 of file model.h.

**10.11.5.59 tstart**

```
double tstart = 0.0 [protected]
```

starting time

Definition at line 1743 of file model.h.

### 10.11.5.60 steadyStateSensitivityMode

```
SteadyStateSensitivityMode steadyStateSensitivityMode = SteadyStateSensitivityMode::newtonOnly
[protected]
```

flag indicating whether steadystate sensitivities are to be computed via FSA when steadyStateSimulation is used

Definition at line 1747 of file model.h.

### 10.11.5.61 reinitializeFixedParameterInitialStates

```
bool reinitializeFixedParameterInitialStates = false [protected]
```

flag indicating whether reinitialization of states depending on fixed parameters is activated

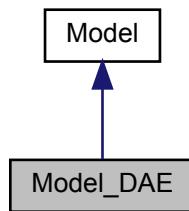
Definition at line 1752 of file model.h.

## 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 fdxdotdp (const realtype t, const N_Vector x, const N_Vector dx)`
- `virtual void fdxdotdp (realtype t, AmiVector *x, AmiVector *dx) 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 *xdot, 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 \*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)
- 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)
- 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 \*dx, 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 \*dx, const realtype \*sx, const realtype \*sdx, const realtype \*w, const realtype \*dwdx, const realtype \*M, const realtype \*J, const realtype \*dxdotdp)
- 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

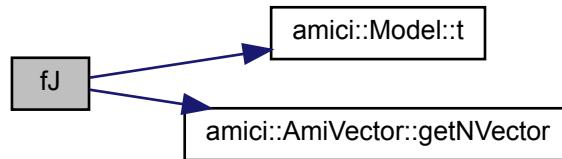
**Parameters**

<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 )
```

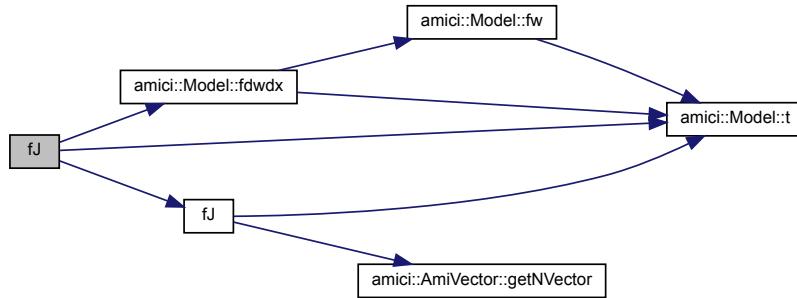
Jacobian of  $x_{dot}$  with respect to states  $x$

#### 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 )
```

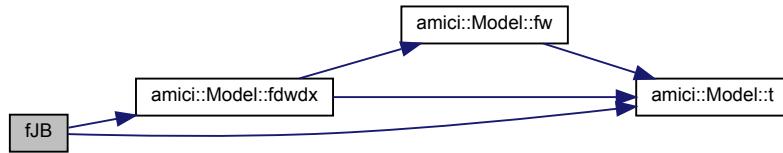
Jacobian of  $\dot{x}_B$  with respect to adjoint state  $x_B$

#### Parameters

<i>t</i>	timepoint
<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>x<sub>B</sub></i>	Vector with the adjoint states
<i>d<sub>x<sub>B</sub></sub></i>	Vector with the adjoint derivative states
<i>JB</i>	Matrix to which the Jacobian will be written

Definition at line 158 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

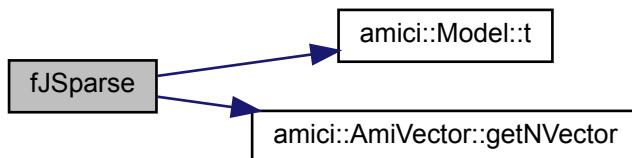
##### Parameters

<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

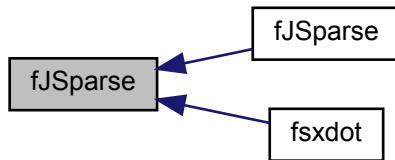
Implements [Model](#).

Definition at line 28 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_Vector x,
    N_Vector dx,
    SlsMat J )
```

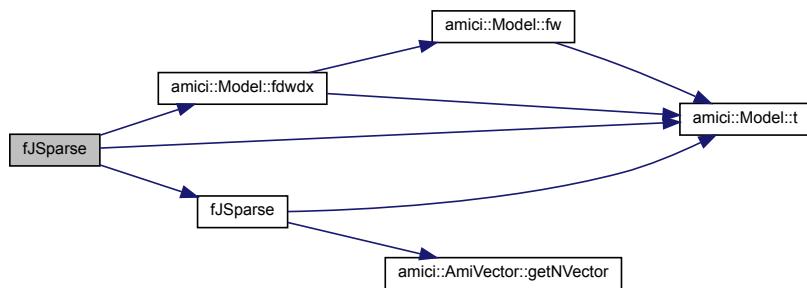
J in sparse form (for sparse solvers from the SuiteSparse Package)

##### Parameters

<code>t</code>	timepoint
<code>cj</code>	scalar in Jacobian (inverse stepsize)
<code>x</code>	Vector with the states
<code>dx</code>	Vector with the derivative states
<code>J</code>	Matrix to which the Jacobian will be written

Definition at line 40 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_Vecor x,
    N_Vecor dx,
    N_Vecor xB,
    N_Vecor dxB,
    SlsMat JB )
```

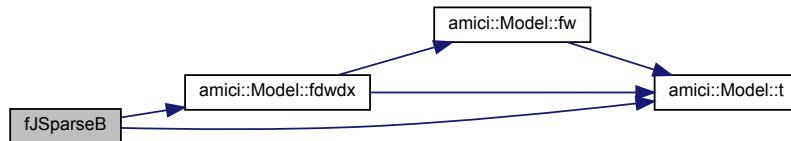
JB in sparse form (for sparse solvers from the SuiteSparse Package)

#### 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 175 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 * JDiag,
    realtype cj,
    AmiVector * x,
    AmiVector * dx ) [override], [virtual]
```

diagonalized Jacobian (for preconditioning)

#### Parameters

<i>t</i>	timepoint
<i>JDiag</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

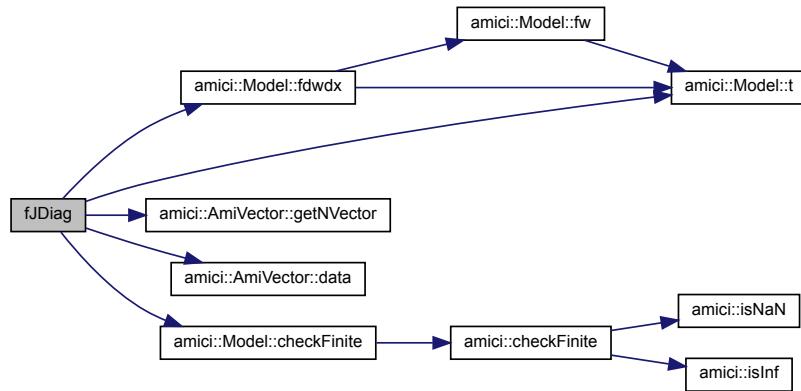
**Returns**

status flag indicating successful execution

Implements [Model](#).

Definition at line 111 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

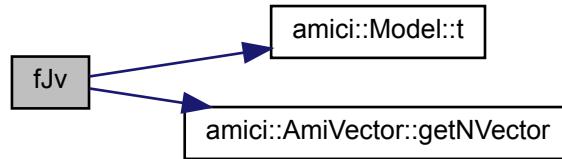
**Parameters**

<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 47 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 )
```

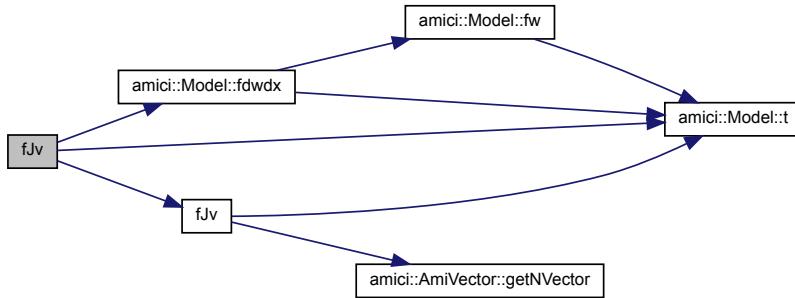
Matrix vector product of J with a vector v (for iterative solvers)

##### Parameters

<i>t</i>	timepoint <b>Type:</b> 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 62 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 )
```

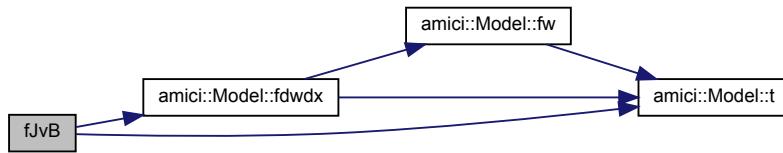
Matrix vector product of JB with a vector v (for iterative solvers)

##### 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 194 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

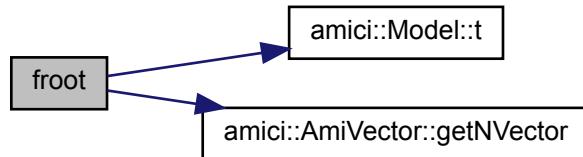
#### Parameters

<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 70 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 )
```

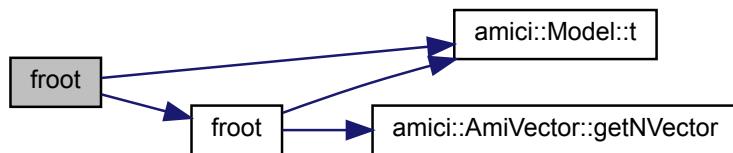
Event trigger function for events

##### 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 80 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

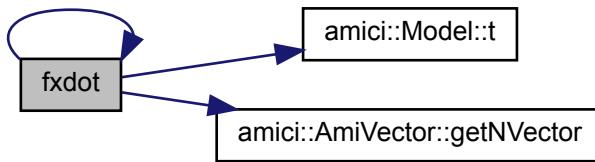
#### Parameters

<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

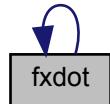
Implements [Model](#).

Definition at line 86 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_Vecor x,
    N_Vecor dx,
    N_Vecor xdot )
```

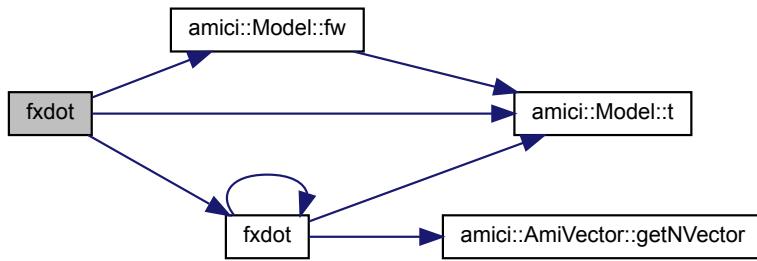
residual function of the DAE

**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 96 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 )
```

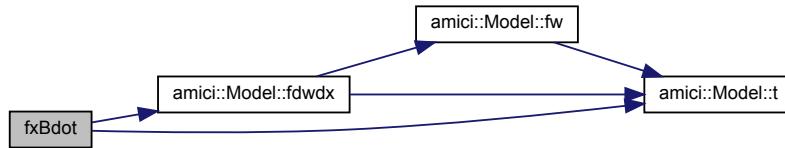
Right hand side of differential equation for adjoint state  $x_B$

**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 211 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_Vector x,
    N_Vector dx,
    N_Vector xB,
    N_Vector dxB,
    N_Vector qBdot )
```

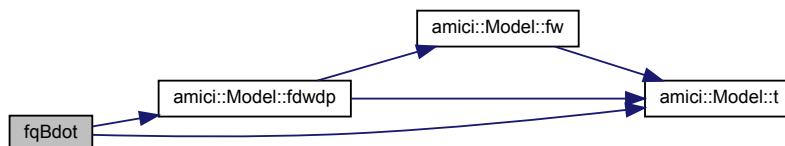
Right hand side of integral equation for quadrature states  $qB$

##### 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 228 of file model\_dae.cpp.

Here is the call graph for this function:



## 10.12.3.17 fwdxotdp() [1/3]

```
void fwdxotdp (
    const realtype t,
    const N_Vector x,
    const N_Vector dx )
```

Sensitivity of  $dx/dt$  wrt model parameters p

**Parameters**

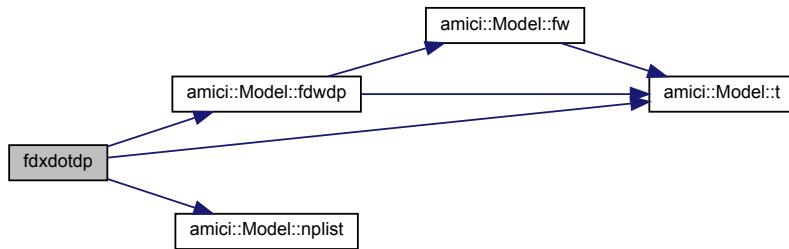
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>dx</i>	Vector with the derivative states

**Returns**

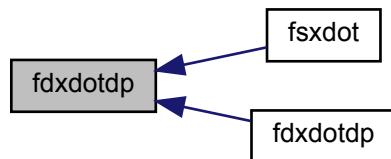
status flag indicating successful execution

Definition at line 127 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 fxdotdp() [2/3]

```
virtual void fxdotdp (
    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)

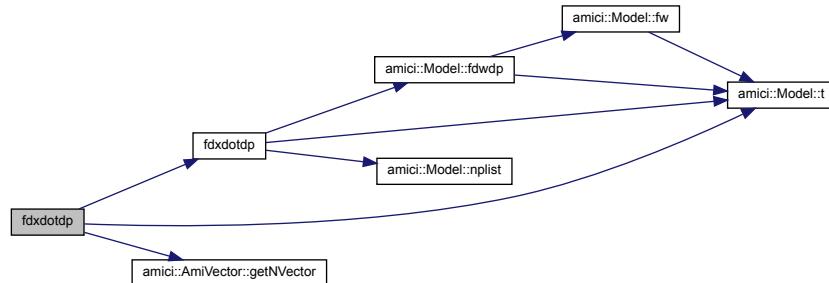
#### Returns

flag indicating successful evaluation

Implements [Model](#).

Definition at line 95 of file `model_dae.h`.

Here is the call graph for this function:



### 10.12.3.19 fsxdot() [1/2]

```
void fsxdot (
    realtype t,
    N_Vecor x,
    N_Vecor dx,
    int ip,
    N_Vecor sx,
    N_Vecor sdx,
    N_Vecor sxdot )
```

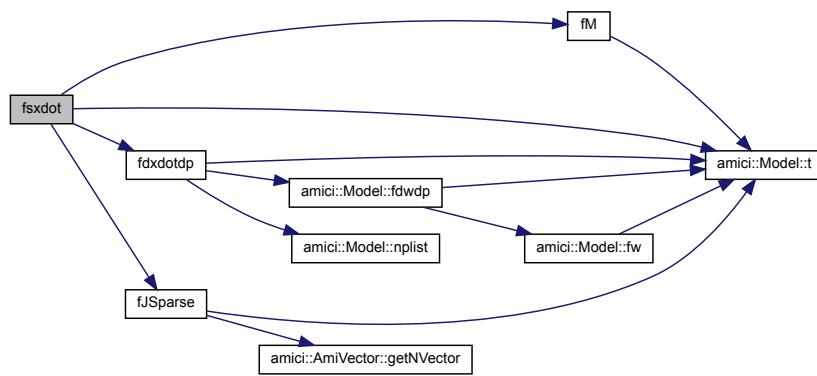
Right hand side of differential equation for state sensitivities sx

**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 247 of file model\_dae.cpp.

Here is the call graph for this function:

**10.12.3.20 fM() [1/2]**

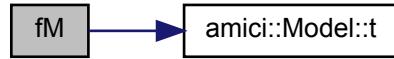
```
void fM (
    realtype t,
    const N_Vector x )
```

**Parameters**

<i>t</i>	timepoint
<i>x</i>	Vector with the states

Definition at line 140 of file model\_dae.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.12.3.21 getSolver()

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

Retrieves the solver object

Returns

The [Solver](#) instance

Implements [Model](#).

Definition at line 145 of file `model_dae.cpp`.

### 10.12.3.22 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>	Vector with the derivative states
<i>w</i>	vector with helper variables
<i>dwdx</i>	derivative of w wrt x

**10.12.3.23 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 137 of file model\_dae.h.

### 10.12.3.24 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

### 10.12.3.25 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

**Parameters**

<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 172 of file model\_dae.h.

**10.12.3.26 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 190 of file model\_dae.h.

**10.12.3.27 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
<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 208 of file model\_dae.h.

#### 10.12.3.28 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

**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>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 <i>w</i> wrt <i>x</i>

Definition at line 229 of file model\_dae.h.

**10.12.3.29 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 244 of file model\_dae.h.

**10.12.3.30 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

#### 10.12.3.31 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 275 of file model\_dae.h.

## 10.12.3.32 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 295 of file model\_dae.h.

## 10.12.3.33 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 313 of file model\_dae.h.

**10.12.3.34 fsxdot() [2/2]**

```
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 335 of file model\_dae.h.

### 10.12.3.35 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

<i>M</i>	mass matrix
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>p</i>	parameter vector
<i>k</i>	constants vector

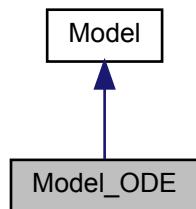
Definition at line 349 of file model\_dae.h.

## 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)`
- `Model_ODE (Model_ODE const &other)`

*Copy constructor.*

  - `virtual void fJ (realtype t, realtype cj, AmiVector **x, AmiVector *dx, AmiVector *xdot, DlsMat J) override`
  - `void fJB (realtype t, N_Vector x, N_Vector xdot, DlsMat J)`
  - `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 *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 (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 fx Bdot (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot)`
  - `void fq Bdot (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, 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 *JDdiag, 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 fx Bdot (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() [1/3]

```
Model_ODE ()
```

default constructor

Definition at line 26 of file `model_ode.h`.

###### 10.13.2.2 Model\_ODE() [2/3]

```
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**

<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 52 of file model\_ode.h.

**10.13.2.3 Model\_ODE() [3/3]**

```
Model_ODE (
    Model_ODE const & other )
```

**Parameters**

<i>other</i>	
--------------	--

Definition at line 66 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

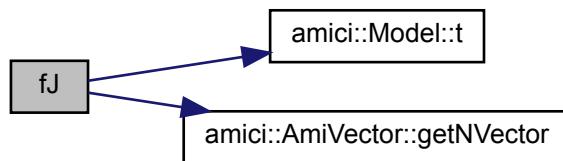
**Parameters**

<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_Vecor x,
    N_Vecor xdot,
    DlsMat J )
  
```

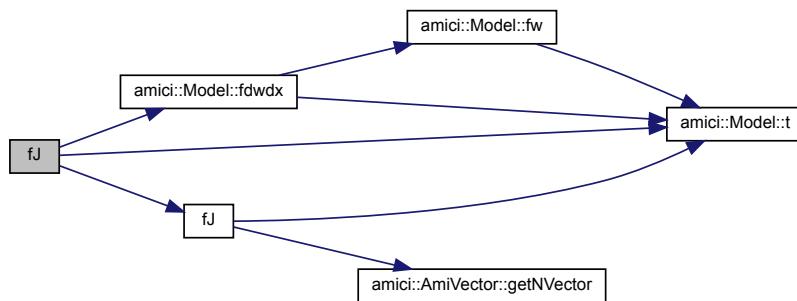
implementation of `fJ` at the `N_Vecor` level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

**Parameters**

$t$	timepoint
$x$	Vector with the 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\_ode.cpp.

Here is the call graph for this function:

**10.13.3.3 fJB() [1/2]**

```
void fJB (
    realtype t,
    N_Vecor x,
    N_Vecor xB,
    N_Vecor xBdot,
    DlsMat JB )
```

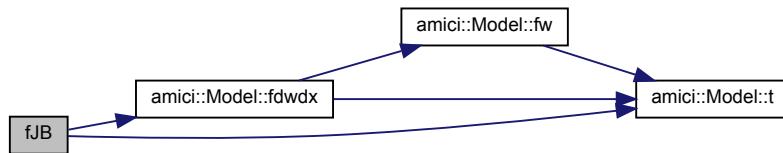
implementation of fJB at the N\_Vecor level, this function provides an interface to the model specific routines for the solver implementation

**Parameters**

$t$	timepoint
$x$	Vector with the states
$xB$	Vector with the adjoint states
$xBdot$	Vector with the adjoint right hand side
$JB$	Matrix to which the Jacobian will be written

Definition at line 141 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

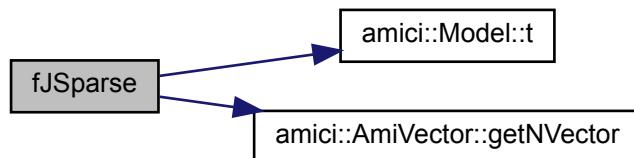
##### Parameters

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

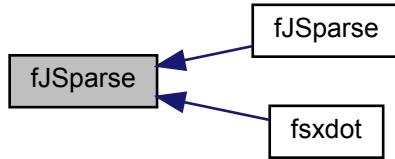
Implements [Model](#).

Definition at line 27 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_Vector x,
    SlsMat J )
```

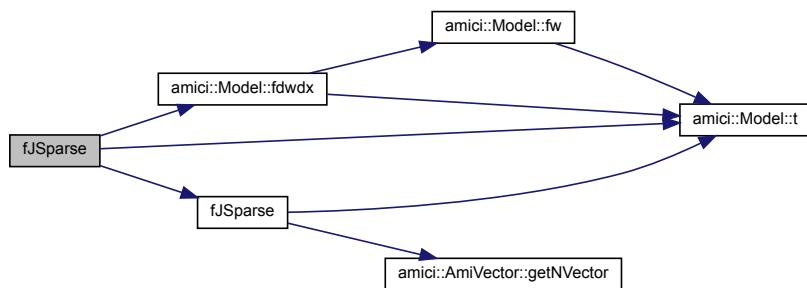
implementation of fJSparse at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

#### Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>J</i>	Matrix to which the Jacobian will be written

Definition at line 39 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_Vector x,
    N_Vector xB,
    N_Vector xBdot,
    SlsMat JB )
```

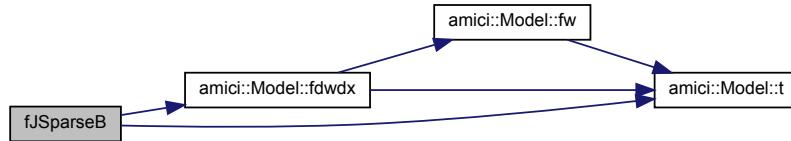
implementation of fJSparseB at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation

## 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 157 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_Vector JDdiag,
    N_Vector x )
```

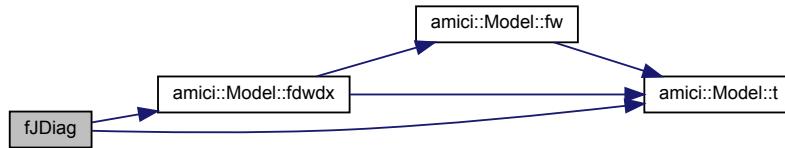
implementation of fJDiag at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation

## Parameters

<i>t</i>	timepoint
<i>JDdiag</i>	Vector to which the Jacobian diagonal will be written
<i>x</i>	Vector with the states

Definition at line 170 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

<code>t</code>	timepoint
<code>JDdiag</code>	Vector to which the Jacobian diagonal will be written
<code>cj</code>	scaling factor, inverse of the step size
<code>x</code>	Vector with the states
<code>dx</code>	Vector with the derivative states

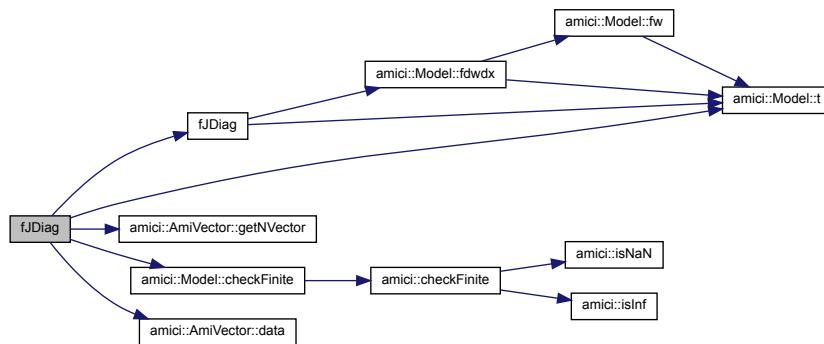
#### Returns

status flag indicating successful execution

Implements [Model](#).

Definition at line 109 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

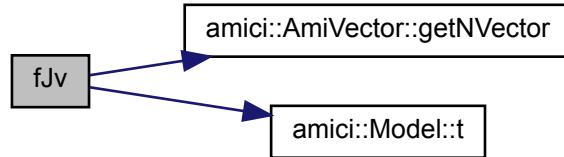
#### Parameters

<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 46 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 )
```

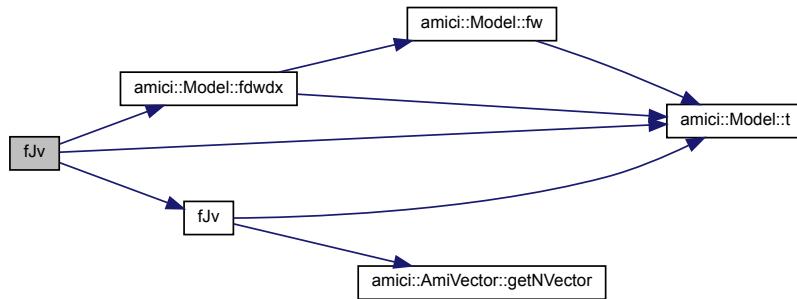
implementation of fJv at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

##### 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 60 of file `model_ode.cpp`.

Here is the call graph for this function:



#### 10.13.3.11 fJvB() [1/2]

```
void fJvB (
    N_Vecotr vB,
    N_Vecotr JvB,
    realtype t,
    N_Vecotr x,
    N_Vecotr xB )
```

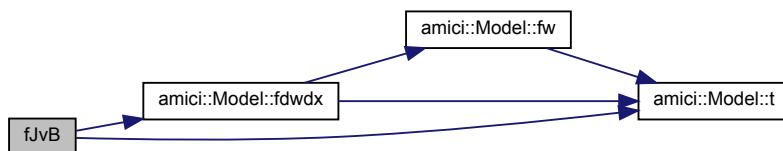
implementation of fJvB at the N\_Vecotr level, this function provides an interface to the model specific routines for the solver implementation

##### 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 186 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

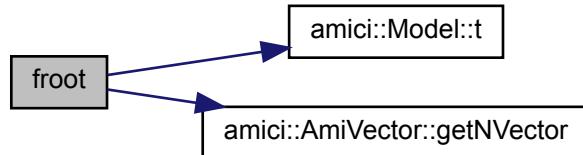
#### Parameters

<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 67 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 )
```

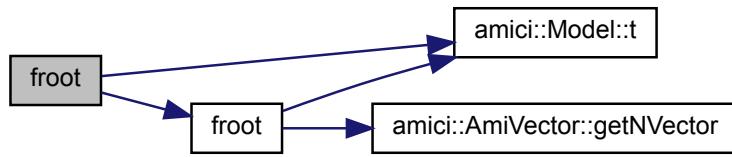
implementation of froot at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

**Parameters**

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>root</i>	array with root function values

Definition at line 78 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

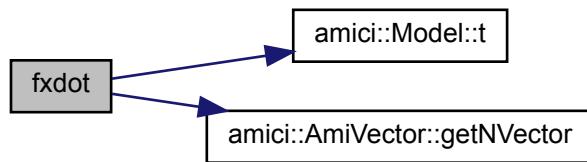
**Parameters**

<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 83 of file model\_ode.cpp.

Here is the call graph for this function:



#### 10.13.3.15 fxdot() [2/3]

```
void fxdot (
    realtype t,
    N_Vector x,
    N_Vector xdot )
```

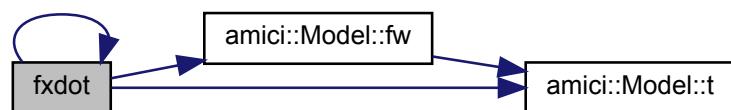
implementation of fxdot at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the [AmiVector](#) level implementation

##### Parameters

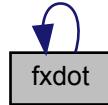
<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>xdot</i>	Vector with the right hand side

Definition at line 94 of file `model_ode.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.13.3.16 fxBdot() [1/2]

```
void fxBdot (
    realtype t,
    N_Vector x,
    N_Vector xB,
    N_Vector xBdot )
```

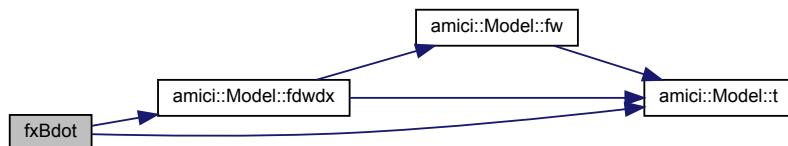
implementation of fxBdot at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation

##### 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

Definition at line 200 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 )
```

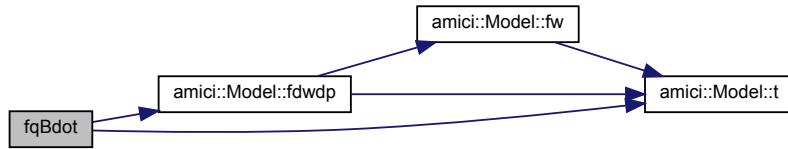
implementation of fqBdot at the N\_Vector level, this function provides an interface to the model specific routines for the solver implementation

## 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 214 of file model\_ode.cpp.

Here is the call graph for this function:



## 10.13.3.18 fdxdotdp() [1/3]

```
void fdxdotdp (
    const realtype t,
    const N_Vector x )
```

Sensitivity of dx/dt wrt model parameters p

## Parameters

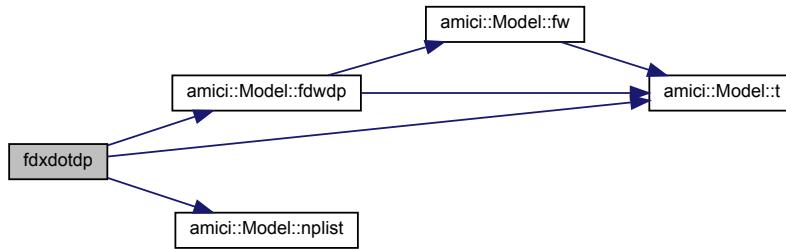
<i>t</i>	timepoint
<i>x</i>	Vector with the states

## Returns

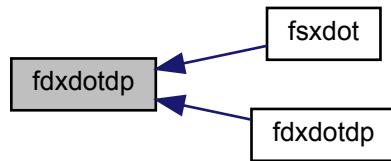
status flag indicating successful execution

Definition at line 121 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)

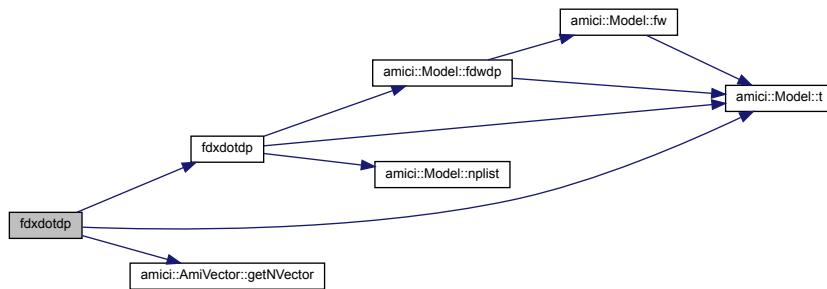
#### Returns

flag indicating successful evaluation

Implements [Model](#).

Definition at line 104 of file model\_ode.h.

Here is the call graph for this function:



#### 10.13.3.20 fsxdot() [1/2]

```
void fsxdot (
    realtype t,
    N_Vecor x,
    int ip,
    N_Vecor sx,
    N_Vecor sxdot )
```

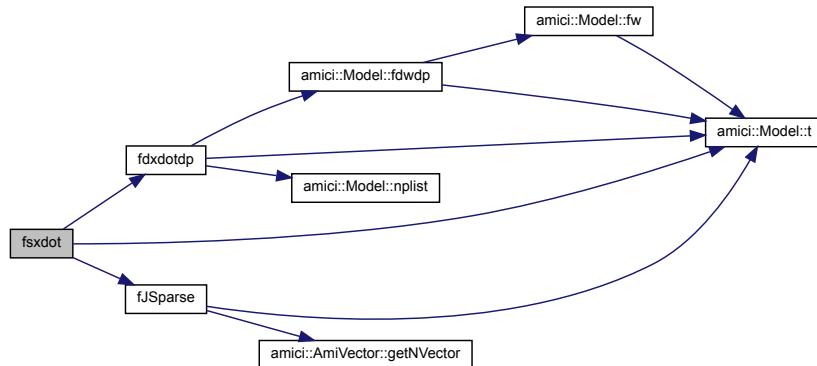
implementation of fsxdot at the N\_Vecor level, this function provides an interface to the model specific routines for the solver implementation

##### Parameters

<i>t</i>	timepoint
<i>x</i>	Vector with the states
<i>ip</i>	parameter index
<i>sx</i>	Vector with the state sensitivities
<i>sxdot</i>	Vector with the sensitivity right hand side

Definition at line 231 of file model\_ode.cpp.

Here is the call graph for this function:



### 10.13.3.21 getSolver()

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

Retrieves the solver object

Returns

The [Solver](#) instance

Implements [Model](#).

Definition at line 129 of file `model_ode.cpp`.

### 10.13.3.22 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 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>w</i>	vector with helper variables

## 10.13.3.23 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 137 of file model\_ode.h.

## 10.13.3.24 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

### 10.13.3.25 fJSparseB() [2/2]

```
virtual void fJSparseB (
    SlsMat JSparseB,
    const realltype t,
    const realltype * x,
    const realltype * p,
    const realltype * k,
    const realltype * h,
    const realltype * xB,
    const realltype * w,
    const realltype * 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 165 of file model\_ode.h.

### 10.13.3.26 fJDiag() [3/3]

```
virtual void fJDiag (
    realltype * JDiag,
    const realltype t,
    const realltype * x,
    const realltype * p,
    const realltype * k,
    const realltype * h,
    const realltype * w,
    const realltype * 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 180 of file model\_ode.h.

#### 10.13.3.27 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 196 of file model\_ode.h.

#### 10.13.3.28 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

**Parameters**

<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 213 of file model\_ode.h.

**10.13.3.29 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 226 of file model\_ode.h.

**10.13.3.30 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

**10.13.3.31 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 252 of file model\_ode.h.

**10.13.3.32 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 269 of file model\_ode.h.

#### 10.13.3.33 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 * 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>w</i>	vector with helper variables
<i>dwdp</i>	derivative of w wrt p

Definition at line 285 of file model\_ode.h.

## 10.13.3.34 fsxdot() [2/2]

```
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

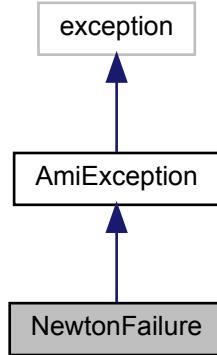
Definition at line 304 of file model\_ode.h.

## 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

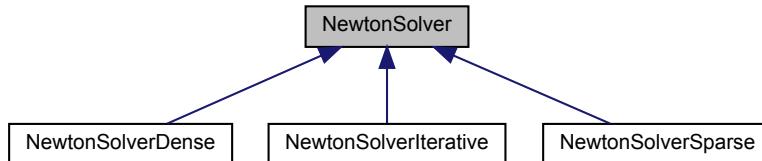
Definition at line 204 of file exception.h.

## 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

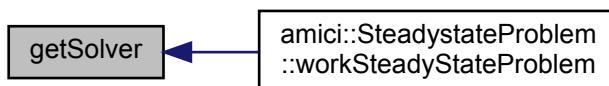
<i>t</i>	pointer to time variable
<i>x</i>	pointer to state variables
<i>linsolType</i>	integer indicating which linear solver to use
<i>model</i>	pointer to the AMICI model object
<i>rdata</i>	pointer to the return data object
<i>maxlinsteps</i>	maximum number of allowed linear steps per Newton step 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 )
```

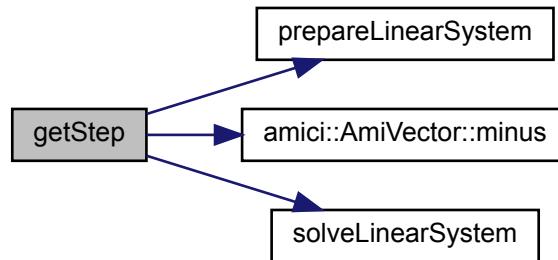
Computes the solution of one Newton iteration

**Parameters**

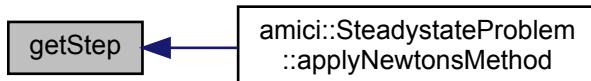
<i>ntry</i>	integer newton_try integer start number of Newton solver (1 or 2)
<i>nnewt</i>	integer number of current Newton step
<i>delta</i>	containing the RHS of the linear system, will be overwritten by solution to the linear system

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 )
```

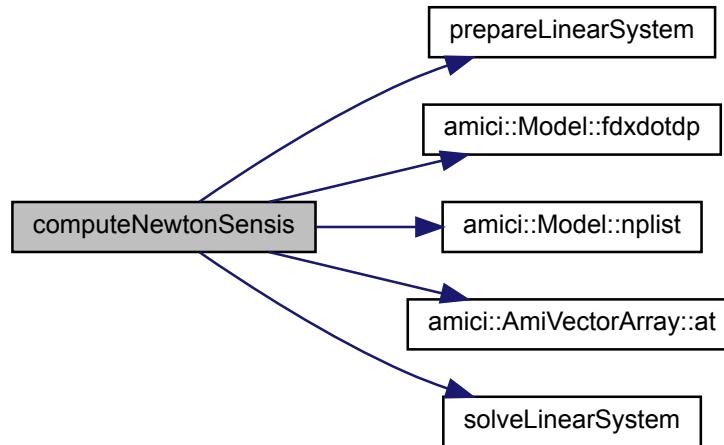
Computes steady state sensitivities

**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]
  
```

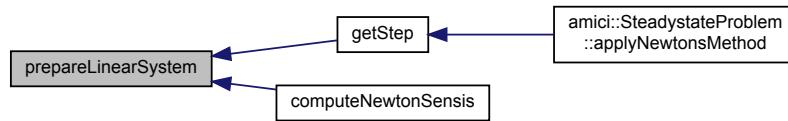
Writes the Jacobian for the Newton iteration and passes it to the linear solver

##### 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]
```

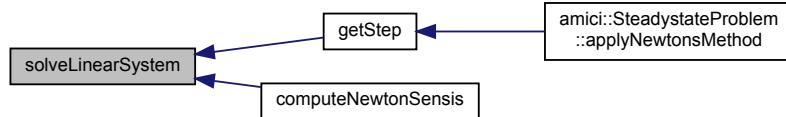
Solves the linear system for the Newton step

#### 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)

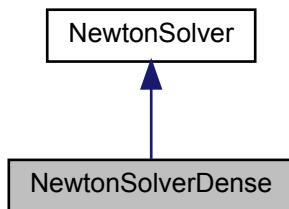
Definition at line 79 of file `newton_solver.h`.

### 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:

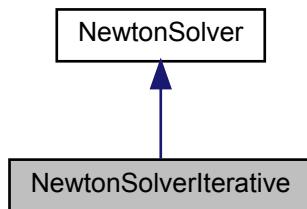


## 10.17 NewtonSolverIterative Class Reference

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

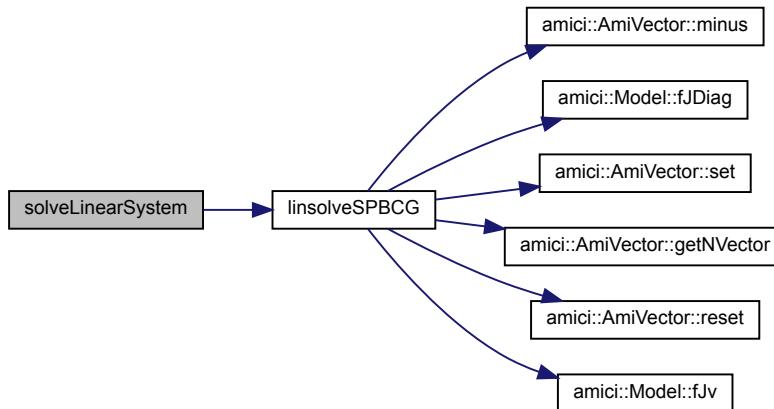
#### Parameters

<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 )
```

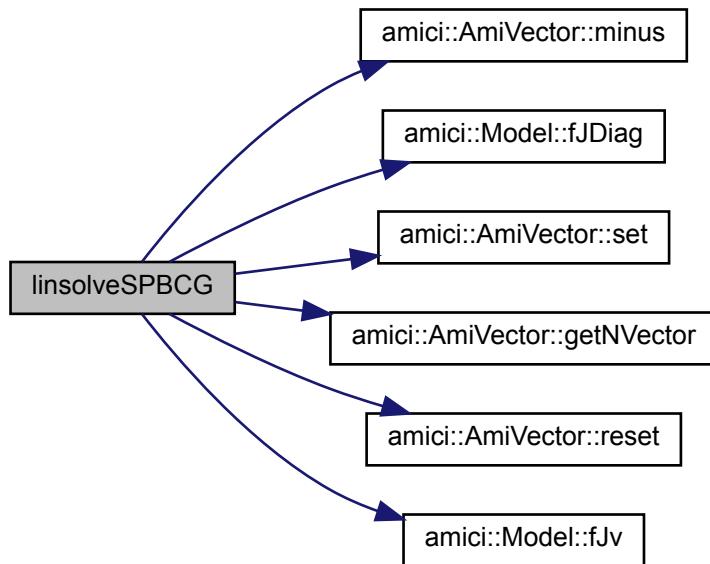
Iterative linear solver created from SPILS BiCG-Stab. Solves the linear system within each Newton step if iterative solver is chosen.

#### 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:

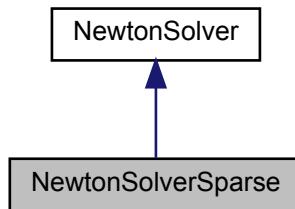


## 10.18 NewtonSolverSparse Class Reference

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 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 \(Solver const &solver, const Model \\*model\)](#)
- [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 > numeritestfails
- std::vector< int > numeritestfailsB
- std::vector< int > numnonlinsolvconvfails
- std::vector< int > numnonlinsolvconvfailsB
- std::vector< int > order
- int newton\_status = 0
- double newton\_time = 0.0
- std::vector< int > newton\_numsteps
- std::vector< int > newton\_numlinsteps
- 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)*

**10.19.1 Detailed Description**

NOTE: multidimensional arrays are stored in row-major order (FORTRAN-style)

Definition at line 28 of file rdata.h.

**10.19.2 Constructor & Destructor Documentation****10.19.2.1 ReturnData()**

```
ReturnData (
    Solver const & solver,
    const Model * model )
```

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

**Parameters**

<i>solver</i>	solver
<i>model</i>	pointer to model specification object bool

Definition at line 22 of file rdata.cpp.

Here is the call graph for this function:

**10.19.3 Member Function Documentation****10.19.3.1 invalidate()**

```
void invalidate (
    const realtype t )
```

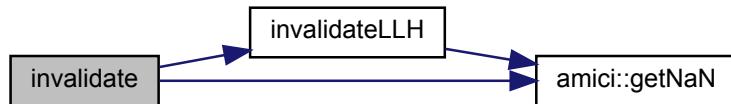
routine to set likelihood, state variables, outputs and respective sensitivities to NaN (typically after integration failure)

**Parameters**

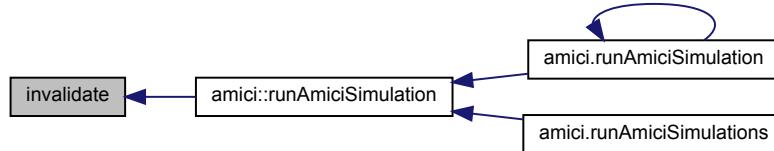
<code>t</code>	time of integration failure
----------------	-----------------------------

Definition at line 105 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.19.3.2 invalidateLLH()**

```
void invalidateLLH ( )
```

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

Definition at line 144 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.19.3.3 applyChainRuleFactorToSimulationResults()

```
void applyChainRuleFactorToSimulationResults (
    const Model * model )
```

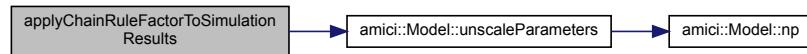
applies the chain rule to account for parameter transformation in the sensitivities of simulation results

#### Parameters

<i>model</i>	Model from which the ReturnData was obtained
--------------	--

Definition at line 156 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.19.4 Friends And Related Function Documentation

#### 10.19.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
    Archive & ar,
    ReturnData & 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

#### 10.19.5 Member Data Documentation

##### 10.19.5.1 ts

```
const std::vector<realtyp> ts
timepoints (dimension: nt)
```

Definition at line 48 of file rdata.h.

##### 10.19.5.2 xdot

```
std::vector<realtyp> xdot
time derivative (dimension: nx)
```

Definition at line 51 of file rdata.h.

##### 10.19.5.3 J

```
std::vector<realtyp> J
Jacobian of differential equation right hand side (dimension: nx x nx, row-major)
```

Definition at line 55 of file rdata.h.

##### 10.19.5.4 z

```
std::vector<realtyp> z
event output (dimension: nmaxevent x nz, row-major)
```

Definition at line 58 of file rdata.h.

**10.19.5.5 sigmaz**

```
std::vector<realtyp> sigmaz
```

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

Definition at line 62 of file rdata.h.

**10.19.5.6 sz**

```
std::vector<realtyp> sz
```

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

Definition at line 66 of file rdata.h.

**10.19.5.7 ssigmaz**

```
std::vector<realtyp> ssigmaz
```

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

Definition at line 70 of file rdata.h.

**10.19.5.8 rz**

```
std::vector<realtyp> rz
```

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

Definition at line 73 of file rdata.h.

**10.19.5.9 srz**

```
std::vector<realtyp> srz
```

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

Definition at line 77 of file rdata.h.

**10.19.5.10 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 81 of file rdata.h.

**10.19.5.11 x**

```
std::vector<realtyp> x
```

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

Definition at line 84 of file rdata.h.

**10.19.5.12 sx**

```
std::vector<realtyp> sx
```

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

Definition at line 88 of file rdata.h.

**10.19.5.13 y**

```
std::vector<realtyp> y
```

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

Definition at line 91 of file rdata.h.

**10.19.5.14 sigmay**

```
std::vector<realtyp> sigmay
```

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

Definition at line 94 of file rdata.h.

**10.19.5.15 sy**

```
std::vector<realtyp> sy
```

parameter derivative of observable (dimension: nt x plist x ny, row-major)

Definition at line 98 of file rdata.h.

**10.19.5.16 ssigmay**

```
std::vector<realtyp> ssigmay
```

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

Definition at line 102 of file rdata.h.

**10.19.5.17 res**

```
std::vector<realtyp> res
```

observable (dimension: nt\*ny, row-major)

Definition at line 105 of file rdata.h.

**10.19.5.18 sres**

```
std::vector<realtyp> sres
```

parameter derivative of residual (dimension: nt\*ny x plist, row-major)

Definition at line 109 of file rdata.h.

**10.19.5.19 FIM**

```
std::vector<realtyp> FIM
```

fisher information matrix (dimension: plist x plist, row-major)

Definition at line 113 of file rdata.h.

**10.19.5.20 numsteps**

```
std::vector<int> numsteps
```

number of integration steps forward problem (dimension: nt)

Definition at line 116 of file rdata.h.

**10.19.5.21 numstepsB**

```
std::vector<int> numstepsB
```

number of integration steps backward problem (dimension: nt)

Definition at line 119 of file rdata.h.

**10.19.5.22 numrhsevals**

```
std::vector<int> numrhsevals
```

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

Definition at line 122 of file rdata.h.

**10.19.5.23 numrhsevalsB**

```
std::vector<int> numrhsevalsB
```

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

Definition at line 125 of file rdata.h.

**10.19.5.24 numerertestfails**

```
std::vector<int> numerertestfails
```

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

Definition at line 128 of file rdata.h.

**10.19.5.25 numerertestfailsB**

```
std::vector<int> numerertestfailsB
```

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

Definition at line 131 of file rdata.h.

**10.19.5.26 numnonlinsolvconvfails**

```
std::vector<int> numnonlinsolvconvfails
```

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

Definition at line 135 of file rdata.h.

**10.19.5.27 numnonlinsolvconvfailsB**

```
std::vector<int> numnonlinsolvconvfailsB
```

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

Definition at line 139 of file rdata.h.

**10.19.5.28 order**

```
std::vector<int> order
```

employed order forward problem (dimension: nt)

Definition at line 142 of file rdata.h.

**10.19.5.29 newton\_status**

```
int newton_status = 0
```

flag indicating success of Newton solver

Definition at line 145 of file rdata.h.

**10.19.5.30 newton\_time**

```
double newton_time = 0.0
```

computation time of the Newton solver [s]

Definition at line 148 of file rdata.h.

**10.19.5.31 newton\_numsteps**

```
std::vector<int> newton_numsteps
```

number of Newton steps for steady state problem (length = 2)

Definition at line 151 of file rdata.h.

**10.19.5.32 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 154 of file rdata.h.

**10.19.5.33 x0**

```
std::vector<realtyp> x0
```

preequilibration steady state found by Newton solver (dimension: nx)

Definition at line 157 of file rdata.h.

**10.19.5.34 sx0**

```
std::vector<realtyp> sx0
```

preequilibration sensitivities found by Newton solver (dimension: nplist x nx, row-major)

Definition at line 160 of file rdata.h.

**10.19.5.35 llh**

```
realtype llh = 0.0
```

loglikelihood value

Definition at line 163 of file rdata.h.

**10.19.5.36 chi2**

```
realtype chi2 = 0.0
```

chi2 value

Definition at line 166 of file rdata.h.

**10.19.5.37 sllh**

```
std::vector<realtype> sllh
```

parameter derivative of loglikelihood (dimension: nplist)

Definition at line 169 of file rdata.h.

**10.19.5.38 s2llh**

```
std::vector<realtype> s2llh
```

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

Definition at line 173 of file rdata.h.

**10.19.5.39 status**

```
int status = 0
```

status code

Definition at line 176 of file rdata.h.

**10.19.5.40 np**

```
const int np
```

total number of model parameters

Definition at line 180 of file rdata.h.

**10.19.5.41 nk**

```
const int nk
```

number of fixed parameters

Definition at line 182 of file rdata.h.

**10.19.5.42 nx**

```
const int nx
```

number of states

Definition at line 184 of file rdata.h.

**10.19.5.43 nxtrue**

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 186 of file rdata.h.

**10.19.5.44 ny**

```
const int ny
```

number of observables

Definition at line 188 of file rdata.h.

**10.19.5.45 nytrue**

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 190 of file rdata.h.

**10.19.5.46 nz**

```
const int nz
```

number of event outputs

Definition at line 192 of file rdata.h.

**10.19.5.47 nztrue**

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 194 of file rdata.h.

**10.19.5.48 ne**

```
const int ne
```

number of events

Definition at line 196 of file rdata.h.

**10.19.5.49 nJ**

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 198 of file rdata.h.

**10.19.5.50 nplist**

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 201 of file rdata.h.

**10.19.5.51 nmaxevent**

```
const int nmaxevent
```

maximal number of occurring events (for every event type)

Definition at line 203 of file rdata.h.

**10.19.5.52 nt**

```
const int nt
```

number of considered timepoints

Definition at line 205 of file rdata.h.

**10.19.5.53 newton\_maxsteps**

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 207 of file rdata.h.

**10.19.5.54 pscale**

```
std::vector<ParameterScaling> pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 209 of file rdata.h.

**10.19.5.55 o2mode**

```
const SecondOrderMode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 211 of file rdata.h.

**10.19.5.56 sensi**

```
const SensitivityOrder sensi
```

sensitivity order

Definition at line 213 of file rdata.h.

**10.19.5.57 sensi\_meth**

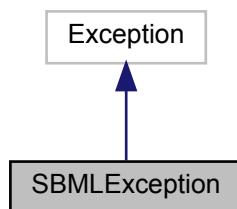
```
const SensitivityMethod sensi_meth
```

sensitivity method

Definition at line 215 of file rdata.h.

**10.20 SBMLError Class Reference**

Inheritance diagram for SBMLError:

**10.20.1 Detailed Description**

Definition at line 19 of file sbml\_import.py.

## 10.21 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 [`loadSBMLFile`](#) (self, SBMLFile)
 

*Parse the provided SBML file.*
- def [`checkLibSBMLErrors`](#) (self)
 

*Generate AMICI C++ files for the model provided to the constructor.*
- 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 [`setName`](#) (self, `modelName`)
 

*Sets the model name.*
- def [`setPaths`](#) (self, `output_dir=None`)
 

*Set output paths for the model and create if necessary.*
- 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)
 

*Get compartment information, stoichiometric matrix and fluxes from SBML model.*
- def [`processReactions`](#) (self)
 

*Get reactions 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 [`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.*
- def [`getSparseSymbols`](#) (self, `symbolName`)
 

*Create sparse symbolic matrix.*
- def [`computeModelEquations`](#) (self, `observables=None`, `sigmas=None`)
 

*Perform symbolic computations required to populate functions in self.functions.*
- def [`computeModelEquationsLinearSolver`](#) (self)
 

*Perform symbolic computations required for the use of various linear solvers.*

- def `computeModelEquationsObjectiveFunction` (self, `observables`=None, `sigmas`=None)  
*Perform symbolic computations required for objective function evaluation.*
- def `computeModelEquationsSensitivitiesCore` (self)  
*Perform symbolic computations required for any sensitivity analysis.*
- def `computeModelEquationsForwardSensitivities` (self)  
*Perform symbolic computations required for forward sensitivity analysis.*
- def `computeModelEquationsAdjointSensitivities` (self)  
*Perform symbolic computations required for adjoint sensitivity analysis.*
- def `prepareModelFolder` (self)  
*Remove all files from the model folder.*
- def `generateCCode` (self, `assume_pow_positivity`)  
*Create C++ code files for the model based on.*
- def `compileCCode` (self, `verbose`=False, `compiler`=None)  
*Compile the generated model code.*
- def `writeIndexFiles` (self, name)  
*Write index file for a symbolic array.*
- def `writeFunctionFile` (self, `function`, `assume_pow_positivity`)  
*Write the function function.*
- def `getFunctionBody` (self, `function`)  
*Generate C++ code for body of function function.*
- def `writeWrapfunctionsCPP` (self)  
*Write model-specific 'wrapper' file (wrapfunctions.cpp).*
- def `writeWrapfunctionsHeader` (self)  
*Write model-specific header file (wrapfunctions.h).*
- def `writeModelHeader` (self)  
*Write model-specific header file (MODELNAME.h).*
- def `getSymbolNameInitializerList` (self, name)  
*Get SBML name initializer list for vector of names for the given model entity.*
- def `getSymbolIDInitializerList` (self, name)  
*Get C++ initializer list for vector of names for the given model entity.*
- def `writeCMakeFile` (self)  
*Write CMake CMakeLists.txt file for this model.*
- def `writeSwigFiles` (self)  
*Write SWIG interface files for this model.*
- def `writeModuleSetup` (self)  
*Create a distutils setup.py file for compile the model module.*
- def `getSymLines` (self, `symbols`, `variable`, `indentLevel`)  
*Generate C++ code for assigning symbolic terms in symbols to C++ array variable.*
- def `getSparseSymLines` (self, `symbolList`, `RowVals`, `ColPtrs`, `variable`, `indentLevel`)  
*Generate C++ code for assigning sparse symbolic matrix to a C++ array variable.*
- def `printWithException` (self, `math`)  
*Generate C++ code for a symbolic expression.*

## Public Attributes

- `allow_reinit_fixpar_initcond`  
*flag indicating whether reinitialization*
- `check_validity`  
*flag indicating whether the validity of the SBML document should be checked*
- `codeprinter`

- codeprinter that allows export of symbolic variables as C++ code*
- **functions**
  - dict carrying function specific definitions*
- **symbols**
  - symbolic definitions*
- **SBMLreader**
  - the libSBML sbml reader [!not storing this will result in a segfault!]*
- **sbml\_doc**
  - document carrying the sbml defintion [!not storing this will result in a segfault!]*
- **sbml**
  - sbml definition [!not storing this will result in a segfault!]*
- **modelName**
  - name of the model that will be used for compilation*
- **modelPath**
  - path to the generated model specific files*
- **modelSwigPath**
  - path to the generated swig files*
- **n\_species**
  - number of species*
- **speciesIndex**
  - dict that maps species names to indices*
- **speciesCompart**
  - array of compartment for each species*
- **constantSpecies**
  - ids of species that are marked as constant*
- **boundaryConditionSpecies**
  - ids of species that are marked as boundary condition*
- **speciesHasOnlySubstanceUnits**
  - array of flags indicating whether a species has only substance units*
- **speciesInitial**
  - array of initial concentrations*
- **speciesConversionFactor**
  - array of conversion factors for every*
- **parameterValues**
  - array of parameter values*
- **n\_parameters**
  - number of parameters*
- **parameterIndex**
  - dict that maps parameter names to indices*
- **fixedParameterValues**
  - array of fixed parameter values*
- **n\_fixed\_parameters**
  - number of fixed parameters*
- **fixedParameterIndex**
  - dict that maps fixed parameter names to indices*
- **compartmentSymbols**
  - array of compartment ids*
- **compartmentVolume**
  - array of compartment volumes*
- **n\_reactions**
  - number of reactions*

- **stoichiometricMatrix**  
*stoichiometry matrix of the model*
- **fluxVector**  
*vector of reaction kinetic laws*
- **observables**  
*array of observable definitions*
- **n\_observables**  
*number of observables*

### 10.21.1 Detailed Description

Definition at line 27 of file sbml\_import.py.

### 10.21.2 Constructor & Destructor Documentation

#### 10.21.2.1 \_\_init\_\_()

```
def __init__ (
    self,
    SBMLFile,
    check_validity = True )
```

##### Parameters

<i>SBMLFile</i>	Path to SBML file where the model is specified
<i>check_validity</i>	Flag indicating whether the validity of the SBML document should be checked

##### Returns

SbmlImporter instance with attached SBML document

Definition at line 141 of file sbml\_import.py.

### 10.21.3 Member Function Documentation

#### 10.21.3.1 loadSBMLFile()

```
def loadSBMLFile (
    self,
    SBMLFile )
```

##### Parameters

<i>SBMLFile</i>	path to SBML file
-----------------	-------------------

**Returns**

Definition at line 303 of file sbml\_import.py.

**10.21.3.2 checkLibSBMLErrors()**

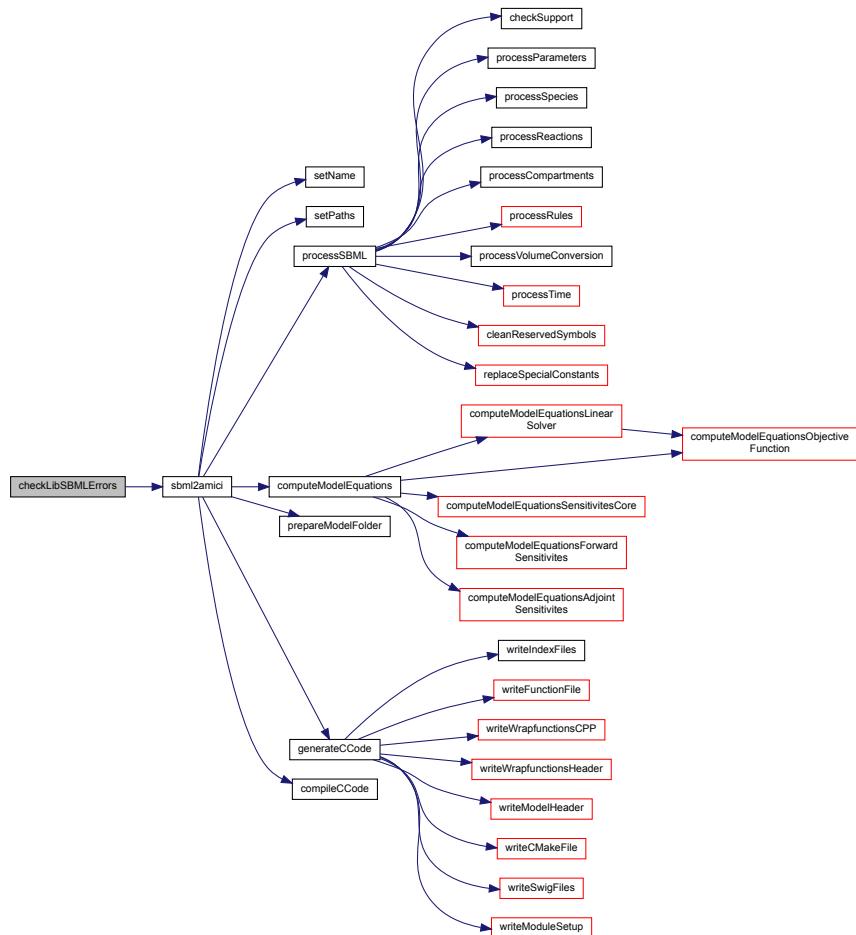
```
def checkLibSBMLErrors (
    self )
```

**Returns**

Checks the error log in the current self.sbml\_doc and raises an exception if errors with severity ERRC or FATAL have occurred

Definition at line 340 of file sbml\_import.py.

Here is the call graph for this function:



### 10.21.3.3 sbml2amici()

```
def sbml2amici (
    self,
    modelName,
    output_dir = None,
    observables = None,
    constantParameters = None,
    sigmas = None,
    verbose = False,
    assume_pow_positivity = False,
    compiler = None )
```

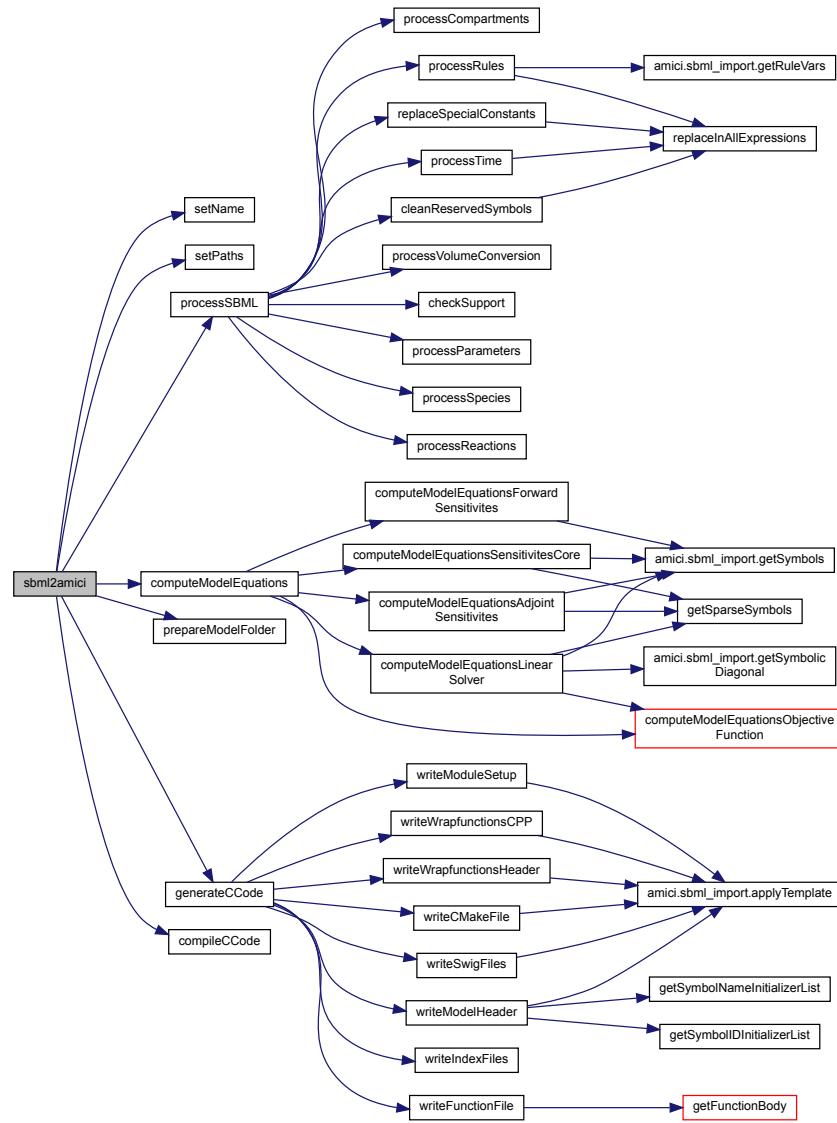
#### Parameters

<i>modelName</i>	name of the model/model directory
<i>output_dir</i>	see <code>sbml_import.setPaths()</code>
<i>observables</i>	dictionary(observableId:{'name':observableName (optional) , 'formula':formulaString}) to be added to the model
<i>sigmas</i>	dictionary(observableId: sigma value or (existing) parameter name)
<i>constantParameters</i>	list of SBML Ids identifying constant parameters
<i>verbose</i>	more verbose output if True
<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
<i>compiler</i>	distutils/setuptools compiler selection to build the python extension

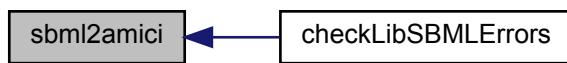
#### Returns

Definition at line 384 of file `sbml_import.py`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.21.3.4 setName()

```
def setName (   
    self,  
    modelName )
```

##### Parameters

<i>modelName</i>	name of the model (must only contain valid filename characters)
------------------	---

##### Returns

Definition at line 413 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.5 setPaths()

```
def setPaths (   
    self,  
    output_dir = None )
```

##### Parameters

<i>output_dir</i>	relative or absolute path where the generated model code is to be placed. will be created if does not exists. defaults to <code>pwd/amici-\$modelName</code> .
-------------------	--

##### Returns

Definition at line 429 of file sbml\_import.py.

Here is the caller graph for this function:



### 10.21.3.6 processSBML()

```
def processSBML (
    self,
    constantParameters = None )
```

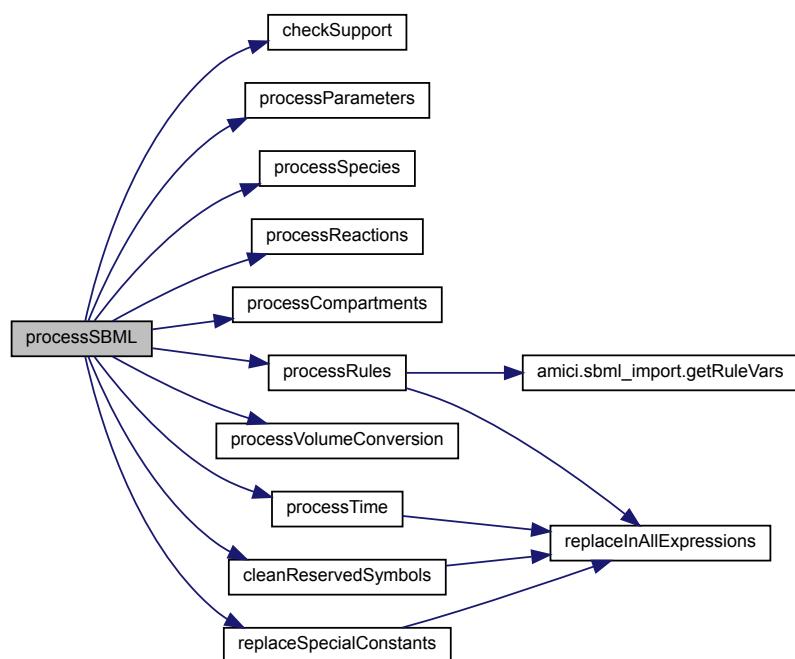
#### Parameters

<code>constantParameters</code>	list of SBML Ids identifying constant parameters
---------------------------------	--

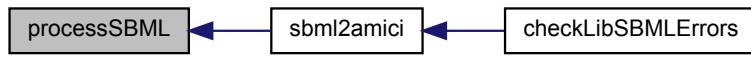
#### Returns

Definition at line 453 of file `sbml_import.py`.

Here is the call graph for this function:



Here is the caller graph for this function:



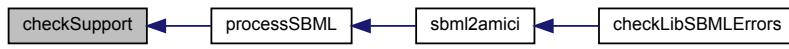
#### 10.21.3.7 checkSupport()

```
def checkSupport ( self )
```

**Returns**

Definition at line 479 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.8 processSpecies()

```
def processSpecies ( self )
```

**Returns**

Definition at line 517 of file sbml\_import.py.

Here is the caller graph for this function:



**10.21.3.9 processParameters()**

```
def processParameters (   
    self,   
    constantParameters = None )
```

**Parameters**

<i>constantParameters</i>	list of SBML Ids identifying constant parameters
---------------------------	--

**Returns**

Definition at line 596 of file sbml\_import.py.

Here is the caller graph for this function:

**10.21.3.10 processCompartments()**

```
def processCompartments (self )
```

**Returns**

Definition at line 652 of file sbml\_import.py.

Here is the caller graph for this function:



### 10.21.3.11 processReactions()

```
def processReactions (
    self )
```

#### Returns

Definition at line 681 of file sbml\_import.py.

Here is the caller graph for this function:



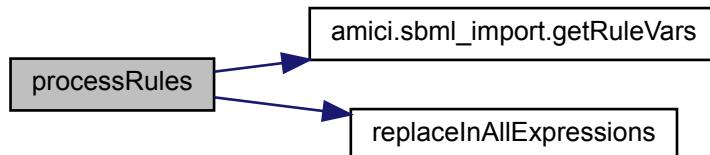
### 10.21.3.12 processRules()

```
def processRules (
    self )
```

#### Returns

Definition at line 774 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.21.3.13 processVolumeConversion()

```
def processVolumeConversion (
    self )
```

##### Returns

Definition at line 849 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.14 processTime()

```
def processTime (
    self )
```

##### Returns

Definition at line 872 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



**10.21.3.15 replaceInAllExpressions()**

```
def replaceInAllExpressions (
    self,
    old,
    new )
```

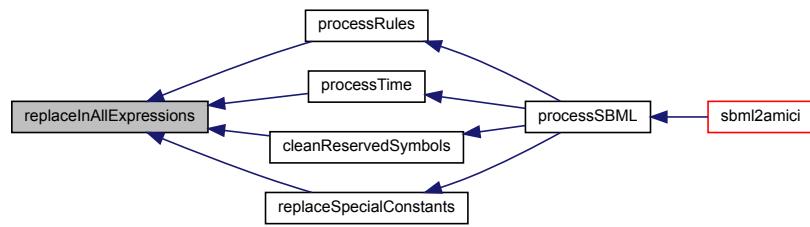
**Parameters**

<i>old</i>	symbolic variables to be replaced
<i>new</i>	replacement symbolic variables

**Returns**

Definition at line 891 of file sbml\_import.py.

Here is the caller graph for this function:

**10.21.3.16 cleanReservedSymbols()**

```
def cleanReservedSymbols (
    self )
```

**Returns**

Definition at line 907 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.17 replaceSpecialConstants()

```
def replaceSpecialConstants (
    self )
```

#### Returns

Definition at line 926 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.18 getSparseSymbols()

```
def getSparseSymbols (
    self,
    symbolName )
```

#### Parameters

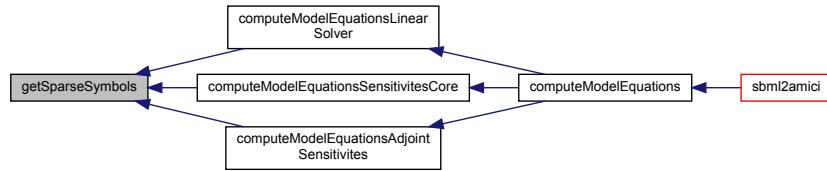
<code>symbolName</code>	name of the function
-------------------------	----------------------

#### Returns

`sparseMatrix` sparse matrix containing symbolic entries  
`symbolList` symbolic vector containing the list of symbol names  
`sparseList` symbolic vector containing the list of symbol formulas  
`symbolColPtrs` Column pointer as specified in the SlsMat definition in CVODES  
`symbolRowVals` Row Values as specified in the SlsMat definition in CVODES

Definition at line 954 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.19 computeModelEquations()

```
def computeModelEquations (
    self,
    observables = None,
    sigmas = None )
```

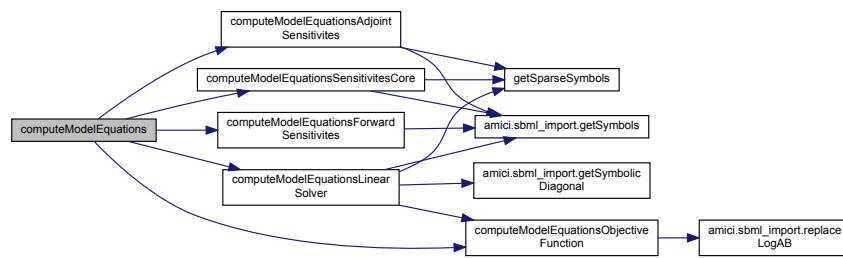
##### Parameters

<i>observables</i>	dictionary(observableId:{'name':observableName (optional) , 'formula':formulaString}) to be added to the model
<i>sigmas</i>	dictionary(observableId: sigma value or (existing) parameter name)

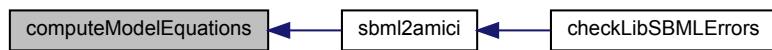
##### Returns

Definition at line 988 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



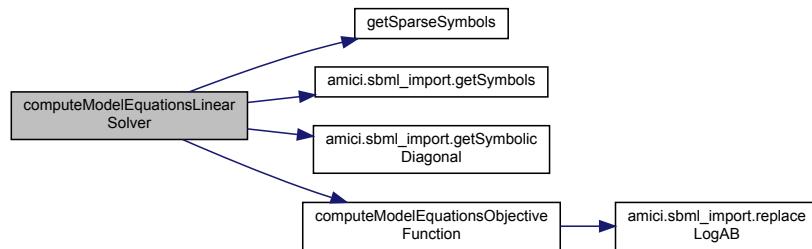
### 10.21.3.20 computeModelEquationsLinearSolver()

```
def computeModelEquationsLinearSolver (
    self )
```

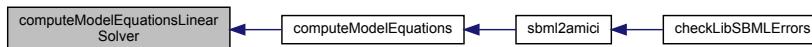
#### Returns

Definition at line 1016 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.21 computeModelEquationsObjectiveFunction()

```
def computeModelEquationsObjectiveFunction (
    self,
    observables = None,
    sigmas = None )
```

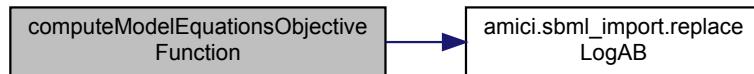
#### Parameters

<code>observables</code>	dictionary(observableId:{'name':observableName (optional) , 'formula':formulaString}) to be added to the model
<code>sigmas</code>	dictionary(observableId: sigma value or (existing) parameter name)

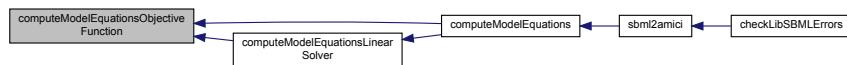
**Returns**

Definition at line 1062 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:

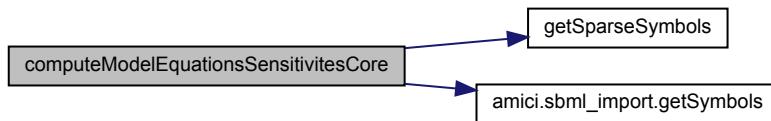
**10.21.3.22 computeModelEquationsSensitivitesCore()**

```
def computeModelEquationsSensitivitesCore (
    self )
```

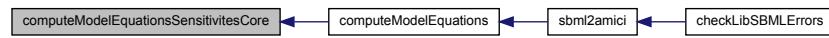
**Returns**

Definition at line 1125 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



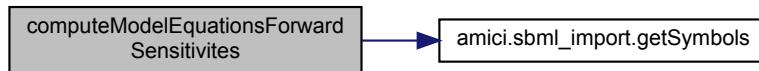
### 10.21.3.23 computeModelEquationsForwardSensitivities()

```
def computeModelEquationsForwardSensitivities (
    self )
```

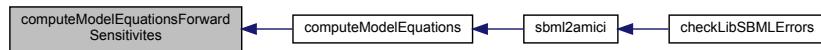
#### Returns

Definition at line 1183 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



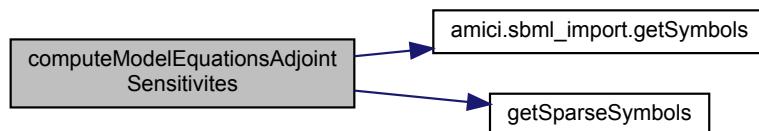
### 10.21.3.24 computeModelEquationsAdjointSensitivities()

```
def computeModelEquationsAdjointSensitivities (
    self )
```

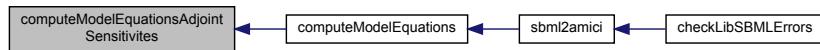
#### Returns

Definition at line 1198 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



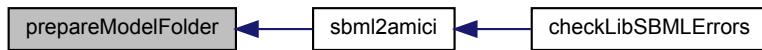
#### 10.21.3.25 prepareModelFolder()

```
def prepareModelFolder ( self )
```

**Returns**

Definition at line 1223 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.26 generateCCode()

```
def generateCCode ( self, assume_pow_positivity )
```

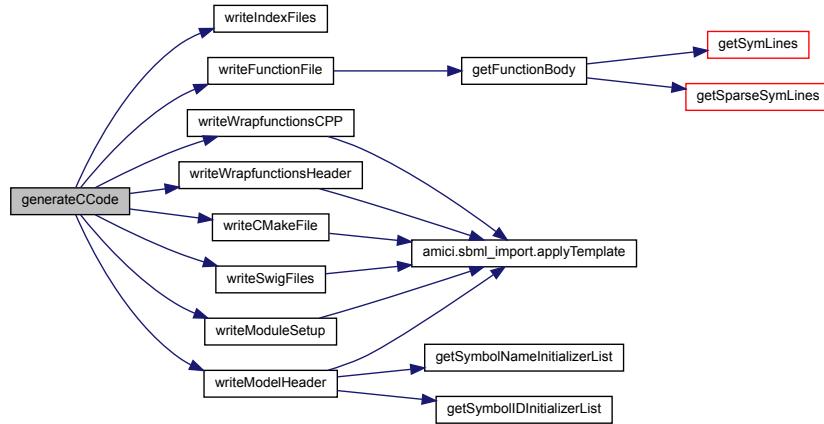
**Parameters**

<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
------------------------------------	--

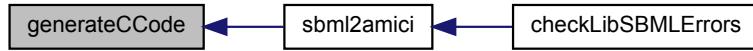
**Returns**

Definition at line 1243 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.27 compileCCode()

```
def compileCCode (
    self,
    verbose = False,
    compiler = None )
```

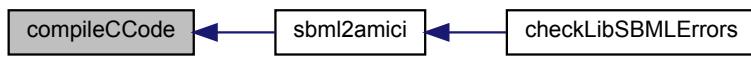
#### Parameters

<i>verbose</i>	Make model compilation verbose
<i>compiler</i>	distutils/setuptools compiler selection to build the python extension

#### Returns

Definition at line 1271 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.28 writeIndexFiles()

```
def writeIndexFiles (
    self,
    name )
```

##### Parameters

<i>name</i>	key in <code>self.symbols</code> for which the respective file should be written
-------------	--

##### Returns

Definition at line 1314 of file `sbml_import.py`.

Here is the caller graph for this function:



#### 10.21.3.29 writeFunctionFile()

```
def writeFunctionFile (
    self,
    function,
    assume_pow_positivity )
```

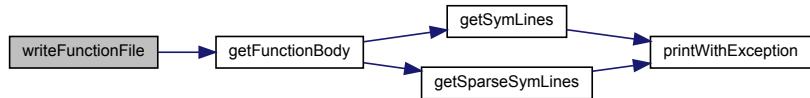
##### Parameters

<i>function</i>	name of the function to be written (see <code>self.functions</code> )
<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

**Returns**

Definition at line 1337 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.30 getFunctionBody()

```
def getFunctionBody (
    self,
    function )
```

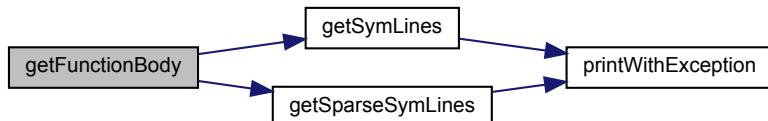
**Parameters**

<i>function</i>	name of the function to be written (see <code>self.functions</code> )
-----------------	---

**Returns**

Definition at line 1395 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.21.3.31 writeWrapfunctionsCPP()

```
def writeWrapfunctionsCPP ( self )
```

**Returns**

Definition at line 1455 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



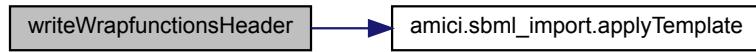
#### 10.21.3.32 writeWrapfunctionsHeader()

```
def writeWrapfunctionsHeader ( self )
```

**Returns**

Definition at line 1470 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:

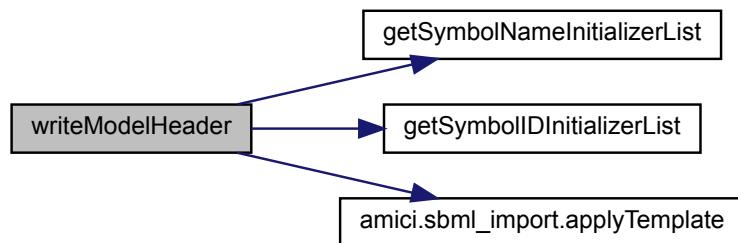
**10.21.3.33 writeModelHeader()**

```
def writeModelHeader ( self )
```

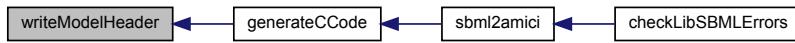
**Returns**

Definition at line 1484 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.21.3.34 getSymbolNameInitializerList()

```
def getSymbolNameInitializerList (
    self,
    name )
```

##### Parameters

<i>name</i>	string, any key present in self.symbols
-------------	---

##### Returns

Template initializer list of names

Definition at line 1548 of file sbml\_import.py.

Here is the caller graph for this function:



#### 10.21.3.35 getSymbolIDInitializerList()

```
def getSymbolIDInitializerList (
    self,
    name )
```

##### Parameters

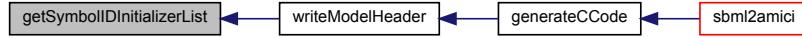
<i>name</i>	string, any key present in self.symbols
-------------	---

##### Returns

Template initializer list of ids

Definition at line 1562 of file sbml\_import.py.

Here is the caller graph for this function:



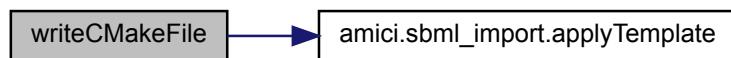
#### 10.21.3.36 writeCMakeFile()

```
def writeCMakeFile (
    self )
```

##### Returns

Definition at line 1574 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



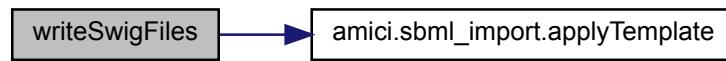
**10.21.3.37 writeSwigFiles()**

```
def writeSwigFiles (
    self )
```

**Returns**

Definition at line 1592 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.21.3.38 writeModuleSetup()**

```
def writeModuleSetup (
    self )
```

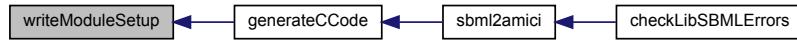
**Returns**

Definition at line 1610 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.21.3.39 getSymLines()

```
def getSymLines (
    self,
    symbols,
    variable,
    indentLevel )
```

#### Parameters

<i>symbols</i>	vectors of symbolic terms
<i>variable</i>	name of the C++ array to assign to
<i>indentLevel</i>	indentation level (number of leading blanks)

#### Returns

C++ code as list of lines

Definition at line 1637 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.21.3.40 getSparseSymLines()

```
def getSparseSymLines (
    self,
    symbolList,
    RowVals,
    ColPtrs,
    variable,
    indentLevel )
```

## Parameters

<i>symbolList</i>	vectors of symbolic terms
<i>RowVals</i>	list of row indices of each nonzero entry (see CVODES SlsMat documentation for details)
<i>ColPtrs</i>	list of indices of the first column entries (see CVODES SlsMat documentation for details)
<i>variable</i>	name of the C++ array to assign to
<i>indentLevel</i>	indentation level (number of leading blanks)

## Returns

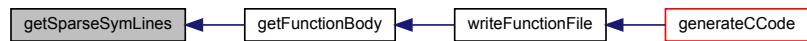
C++ code as list of lines

Definition at line 1667 of file sbml\_import.py.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.21.3.41 printWithException()

```
def printWithException (
    self,
    math )
```

**Parameters**

<i>math</i>	symbolic expression
-------------	---------------------

**Returns**

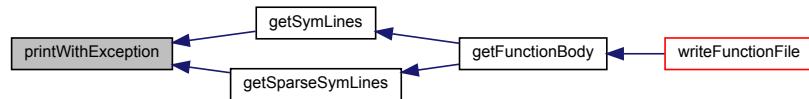
C++ code for the specified expression

**Exceptions**

<i>SBMLError</i>	The specified expression contained an unsupported function
------------------	--

Definition at line 1690 of file sbml\_import.py.

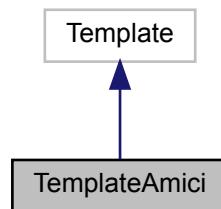
Here is the caller graph for this function:



## 10.22 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*

### 10.22.1 Detailed Description

Returns

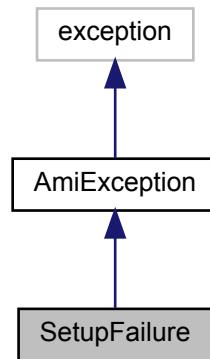
Definition at line 1775 of file sbml\_import.py.

## 10.23 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)

### 10.23.1 Detailed Description

Definition at line 188 of file exception.h.

### 10.23.2 Constructor & Destructor Documentation

#### 10.23.2.1 SetupFailure()

```
SetupFailure (
    const char * msg )
```

constructor, simply calls [AmiException](#) constructor

## Parameters

<code>msg</code>	<input type="text"/>
------------------	----------------------

Definition at line 193 of file exception.h.

## 10.24 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)`  
`setupAMIs initialises the ami memory object`
- `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 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*
- `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)*
- `int getStabilityLimitFlag () const`  
        *returns stability limit detection mode*
- `void setStabilityLimitFlag (int 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 \*numerrtestfails) 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.*

### 10.24.1 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`.

### 10.24.2 Constructor & Destructor Documentation

#### 10.24.2.1 Solver()

```
Solver (
    const Solver & other )
```

##### Parameters

<code>other</code>	<input type="button" value=""/>
--------------------	---------------------------------

Definition at line 46 of file `solver.h`.

### 10.24.3 Member Function Documentation

#### 10.24.3.1 clone()

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

##### Returns

The clone

#### 10.24.3.2 setup()

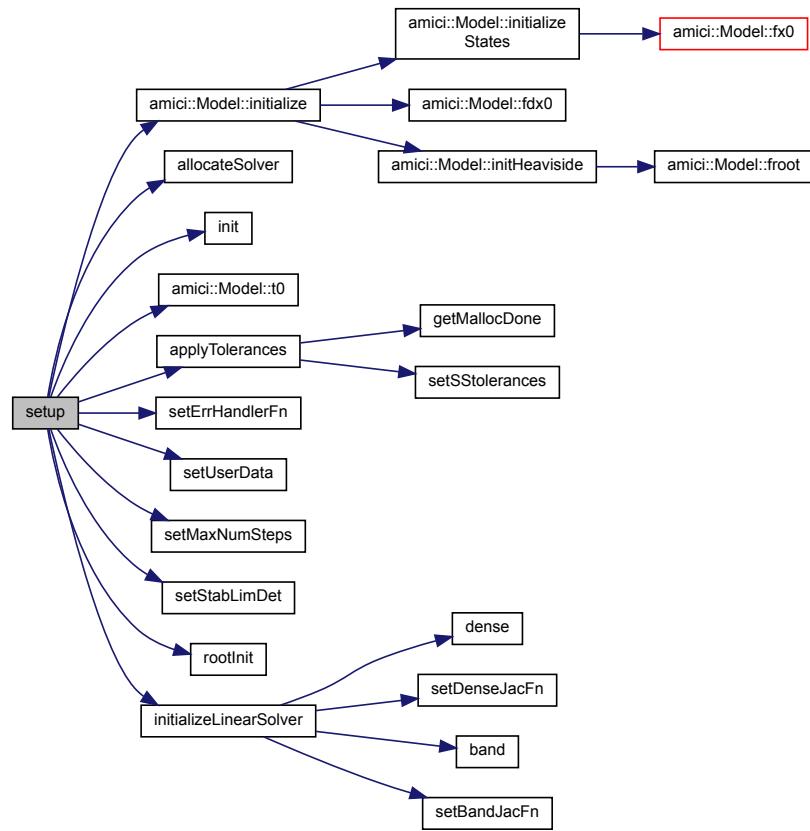
```
void setup (
    AmiVector * x,
    AmiVector * dx,
    AmiVectorArray * sx,
    AmiVectorArray * sdx,
    Model * model )
```

##### Parameters

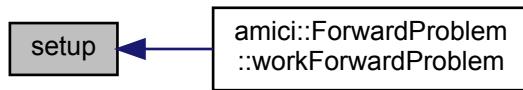
<code>x</code>	state vector
<code>dx</code>	state derivative vector (DAE only)
<code>sx</code>	state sensitivity vector
<code>sdx</code>	state derivative sensitivity vector (DAE only)
<code>model</code>	pointer to the model object

Definition at line 27 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.24.3.3 setupAMIB()

```
void setupAMIB (
    BackwardProblem * bwd,
    Model * model )
```

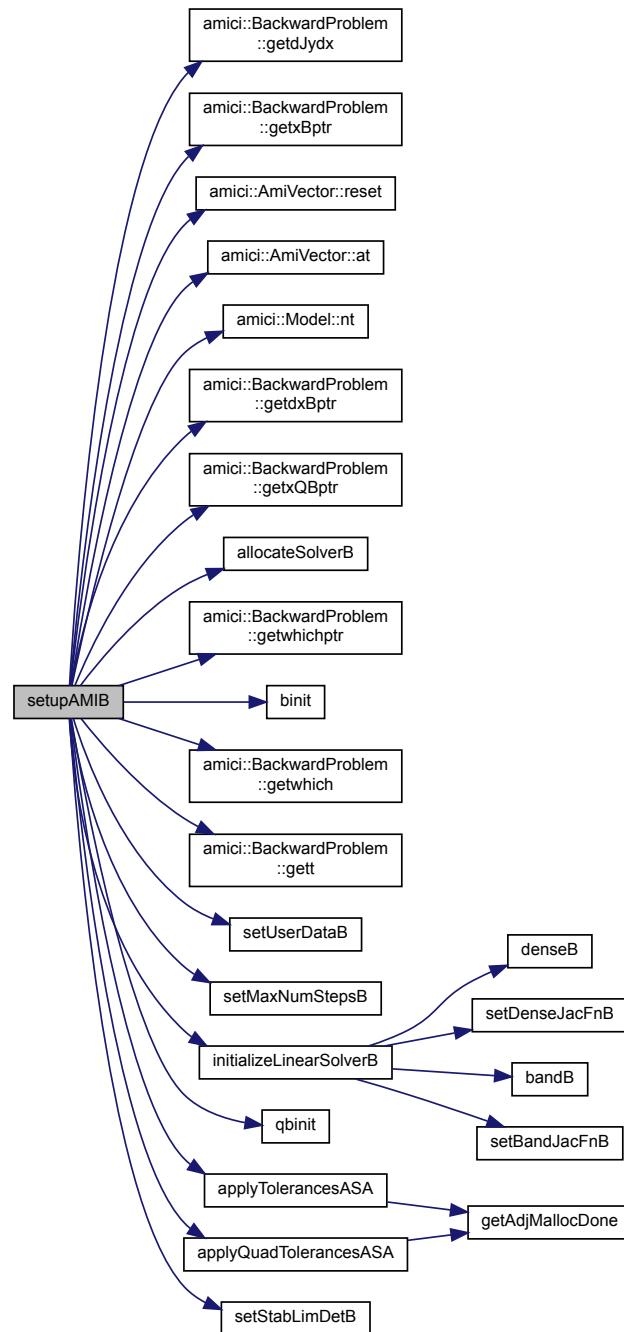
setupAMIB initialises the AMI memory object for the backwards problem

**Parameters**

<i>bwd</i>	pointer to backward problem
<i>model</i>	pointer to the model object

Definition at line 100 of file solver.cpp.

Here is the call graph for this function:



#### 10.24.3.4 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

#### 10.24.3.5 getDiagnosis()

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

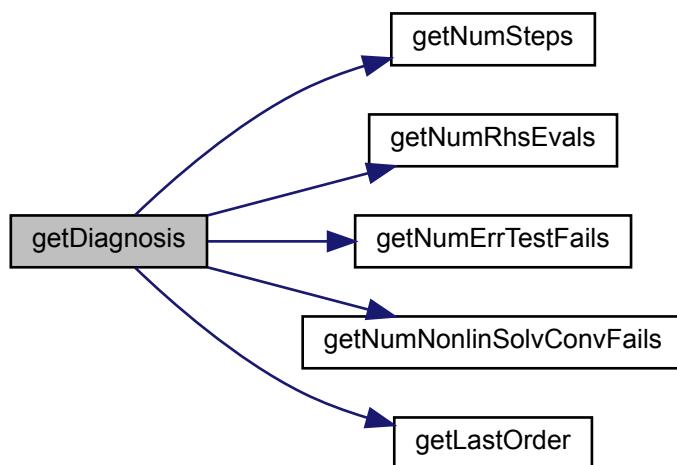
getDiagnosis extracts diagnosis information from solver memory block and writes them into the return data object

##### 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:



### 10.24.3.6 getDiagnosisB()

```
void getDiagnosisB (
    const int it,
    ReturnData * rdata,
    int which ) const
```

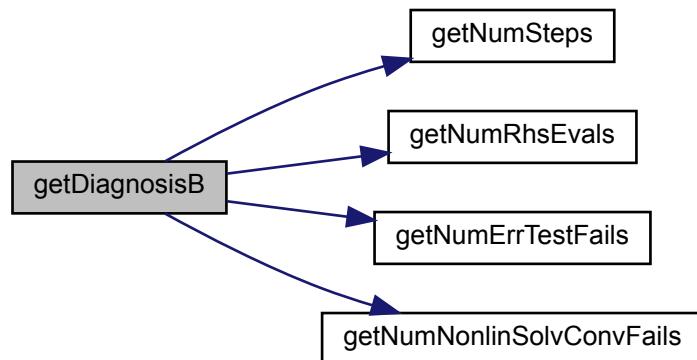
getDiagnosisB extracts diagnosis information from solver memory block and writes them into the return data object for the backward problem

#### 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:



### 10.24.3.7 getRootInfo()

```
virtual void getRootInfo (
    int * rootsfound ) const [pure virtual]
```

getRootInfo extracts information which event occurred

#### Parameters

<i>rootsfound</i>	array with flags indicating whether the respective event occurred
-------------------	---

**10.24.3.8 reInit()**

```
virtual void reInit (
    realtype t0,
    AmiVector * yy0,
    AmiVector * yp0 ) [pure virtual]
```

ReInit reinitializes the states in the solver after an event occurrence

**Parameters**

<i>t0</i>	new timepoint
<i>yy0</i>	new state variables
<i>yp0</i>	new derivative state variables (DAE only)

**10.24.3.9 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)

**10.24.3.10 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)

**10.24.3.11 calcICB()**

```
virtual void calcICB (
    int which,
```

```
realtype tout1,
AmiVector * xB,
AmiVector * dxB ) [pure virtual]
```

CalcIBC 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)

#### 10.24.3.12 solve()

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

Solve solves the forward problem until a predefined timepoint

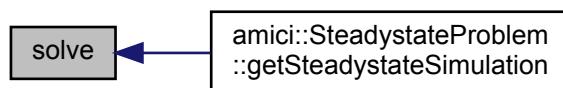
#### Parameters

<i>tout</i>	timepoint until which simulation should be performed
<i>yret</i>	states
<i>ypret</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:



## 10.24.3.13 solveF()

```
virtual int solveF (
    realtype tout,
    AmiVector * yret,
    AmiVector * ypret,
    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>ypret</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

## 10.24.3.14 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

## 10.24.3.15 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
--------------	--

### 10.24.3.16 reInitB()

```
virtual void reInitB (
    int which,
    realtype tB0,
    AmiVector * yyB0,
    AmiVector * ypB0 ) [pure virtual]
```

ReInitB reinitializes the adjoint states after an event occurrence

#### 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)

### 10.24.3.17 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)

### 10.24.3.18 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

### 10.24.3.19 quadReInitB()

```
virtual void quadReInitB (
    int which,
    AmiVector * yQB0 ) [pure virtual]
```

ReInitB reinitializes the adjoint states after an event occurrence

#### Parameters

<i>which</i>	identifier of the backwards problem
<i>yQB0</i>	new adjoint quadrature state variables

### 10.24.3.20 turnOffRootFinding()

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

turnOffRootFinding disables rootfinding

### 10.24.3.21 getSensitivityMethod()

```
SensitivityMethod getSensitivityMethod () const
```

sensitivity method

#### Returns

method enum

Definition at line 466 of file solver.cpp.

Here is the caller graph for this function:



### 10.24.3.22 setSensitivityMethod()

```
void setSensitivityMethod (
    SensitivityMethod sensi_meth )
```

**Parameters**

*sensi\_meth*

Definition at line 470 of file solver.cpp.

Here is the caller graph for this function:

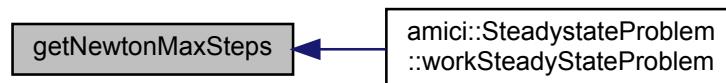
**10.24.3.23 getNewtonMaxSteps()**

```
int getNewtonMaxSteps ( ) const
```

**Returns**

Definition at line 474 of file solver.cpp.

Here is the caller graph for this function:

**10.24.3.24 setNewtonMaxSteps()**

```
void setNewtonMaxSteps ( int newton_maxsteps )
```

**Parameters**

*newton\_maxsteps*

Definition at line 478 of file solver.cpp.

Here is the caller graph for this function:



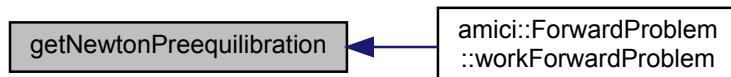
#### 10.24.3.25 getNewtonPreequilibration()

```
bool getNewtonPreequilibration ( ) const
```

**Returns**

Definition at line 484 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.26 setNewtonPreequilibration()

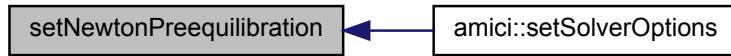
```
void setNewtonPreequilibration (
    bool newton_preeq )
```

**Parameters**

<i>newton_preeq</i>	
---------------------	--

Definition at line 488 of file solver.cpp.

Here is the caller graph for this function:



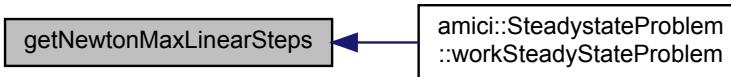
#### 10.24.3.27 getNewtonMaxLinearSteps()

```
int getNewtonMaxLinearSteps ( ) const
```

##### Returns

Definition at line 492 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.28 setNewtonMaxLinearSteps()

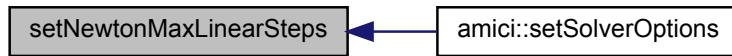
```
void setNewtonMaxLinearSteps (
    int newton_maxlinsteps )
```

##### Parameters

<i>newton_maxlinsteps</i>
---------------------------

Definition at line 496 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.29 getSensitivityOrder()

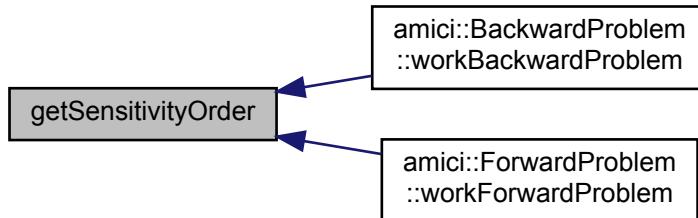
```
SensitivityOrder getSensitivityOrder ( ) const
```

**Returns**

sensitivity order

Definition at line 502 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.30 setSensitivityOrder()

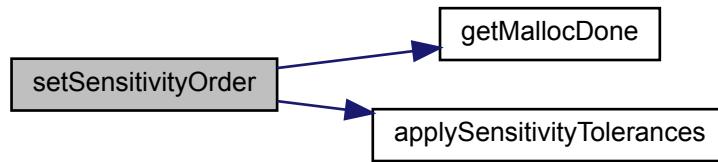
```
void setSensitivityOrder (  
    SensitivityOrder sensi )
```

**Parameters**

<i>sensi</i>	sensitivity order
--------------	-------------------

Definition at line 506 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.24.3.31 `getRelativeTolerance()`

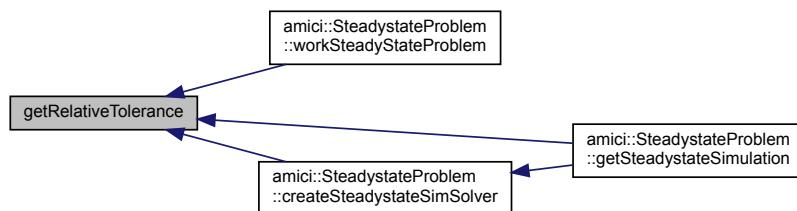
```
double getRelativeTolerance( ) const
```

##### Returns

relative tolerances

Definition at line 513 of file solver.cpp.

Here is the caller graph for this function:



**10.24.3.32 setRelativeTolerance()**

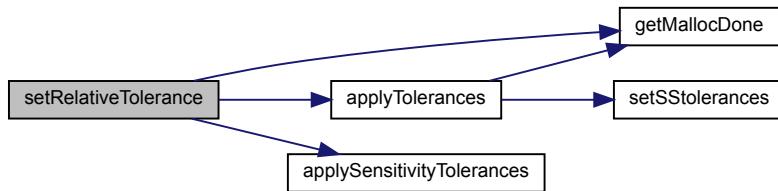
```
void setRelativeTolerance (
    double rtol )
```

**Parameters**

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 517 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.24.3.33 getAbsoluteTolerance()**

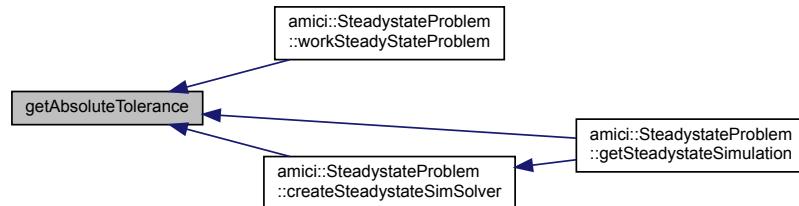
```
double getAbsoluteTolerance ( ) const
```

**Returns**

absolute tolerances

Definition at line 529 of file solver.cpp.

Here is the caller graph for this function:



## 10.24.3.34 setAbsoluteTolerance()

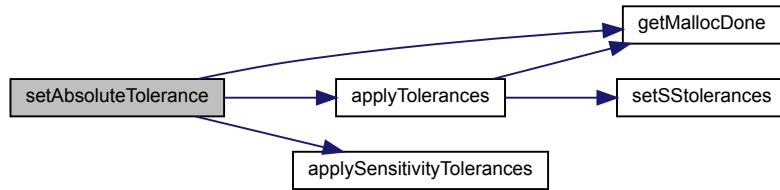
```
void setAbsoluteTolerance (
    double atol )
```

## Parameters

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 533 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.24.3.35 getRelativeToleranceQuadratures()

```
double getRelativeToleranceQuadratures ( ) const
```

## Returns

relative tolerance

Definition at line 545 of file solver.cpp.

## 10.24.3.36 setRelativeToleranceQuadratures()

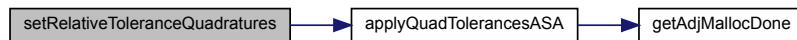
```
void setRelativeToleranceQuadratures (
    double rtol )
```

**Parameters**

<i>rtol</i>	relative tolerance (non-negative number)
-------------	--

Definition at line 549 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.24.3.37 getAbsoluteToleranceQuadratures()**

```
double getAbsoluteToleranceQuadratures ( ) const
```

**Returns**

absolute tolerance

Definition at line 563 of file solver.cpp.

**10.24.3.38 setAbsoluteToleranceQuadratures()**

```
void setAbsoluteToleranceQuadratures (
    double atol )
```

**Parameters**

<i>atol</i>	absolute tolerance (non-negative number)
-------------	--

Definition at line 567 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.24.3.39 getMaxSteps()

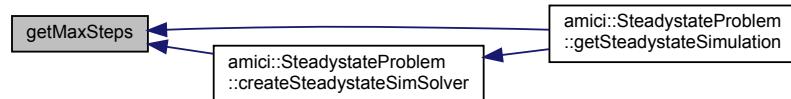
```
int getMaxSteps ( ) const
```

**Returns**

maximum number of solver steps

Definition at line 581 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.40 setMaxSteps()

```
void setMaxSteps ( int maxsteps )
```

**Parameters**

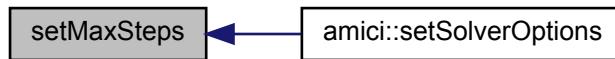
<i>maxsteps</i>	maximum number of solver steps (non-negative number)
-----------------	--

Definition at line 585 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.24.3.41 getMaxStepsBackwardProblem()**

```
int getMaxStepsBackwardProblem ( ) const
```

**Returns**

maximum number of solver steps

Definition at line 594 of file solver.cpp.

**10.24.3.42 setMaxStepsBackwardProblem()**

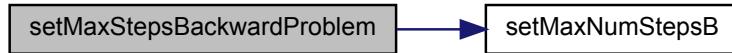
```
void setMaxStepsBackwardProblem ( int maxsteps )
```

**Parameters**

<i>maxsteps</i>	maximum number of solver steps (non-negative number)
-----------------	--

Definition at line 598 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.24.3.43 getLinearMultistepMethod()

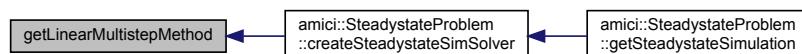
```
LinearMultistepMethod getLinearMultistepMethod ( ) const
```

**Returns**

linear system multistep method

Definition at line 608 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.44 setLinearMultistepMethod()

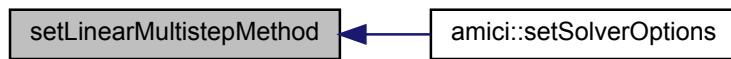
```
void setLinearMultistepMethod ( LinearMultistepMethod lmm )
```

**Parameters**

<i>lmm</i>	linear system multistep method
------------	--------------------------------

Definition at line 612 of file solver.cpp.

Here is the caller graph for this function:

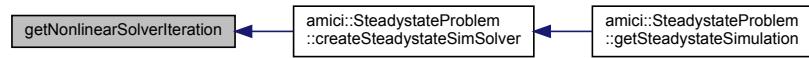
**10.24.3.45 getNonlinearSolverIteration()**

`NonlinearSolverIteration getNonlinearSolverIteration ( ) const`

**Returns**

Definition at line 618 of file solver.cpp.

Here is the caller graph for this function:

**10.24.3.46 setNonlinearSolverIteration()**

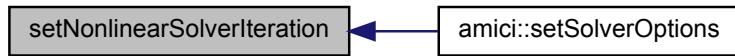
`void setNonlinearSolverIteration ( NonlinearSolverIteration iter )`

**Parameters**

<i>iter</i>	nonlinear system solution method
-------------	----------------------------------

Definition at line 622 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.47 getInterpolationType()

```
InterpolationType getInterpolationType ( ) const
```

##### Returns

Definition at line 628 of file solver.cpp.

#### 10.24.3.48 setInterpolationType()

```
void setInterpolationType ( InterpolationType interpType )
```

##### Parameters

<i>interpType</i>	interpolation type
-------------------	--------------------

Definition at line 632 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.49 getStateOrdering()

```
StateOrdering getStateOrdering ( ) const
```

##### Returns

Definition at line 638 of file solver.cpp.

#### 10.24.3.50 setStateOrdering()

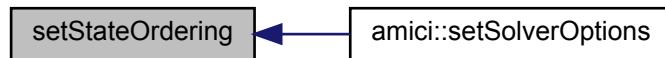
```
void setStateOrdering (
    StateOrdering ordering )
```

##### Parameters

<i>ordering</i>	state ordering
-----------------	----------------

Definition at line 642 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.51 getStabilityLimitFlag()

```
int getStabilityLimitFlag ( ) const
```

##### Returns

std::det can be amici.FALSE (deactivated) or amici.TRUE (activated)

Definition at line 652 of file solver.cpp.

Here is the caller graph for this function:



**10.24.3.52 setStabilityLimitFlag()**

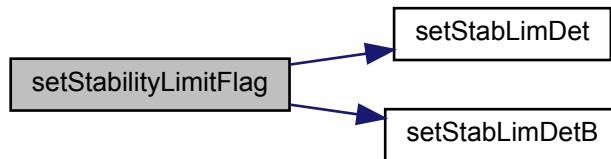
```
void setStabilityLimitFlag ( int stldet )
```

**Parameters**

<i>stldet</i>	can be amici.FALSE (deactivated) or amici.TRUE (activated)
---------------	--

Definition at line 656 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

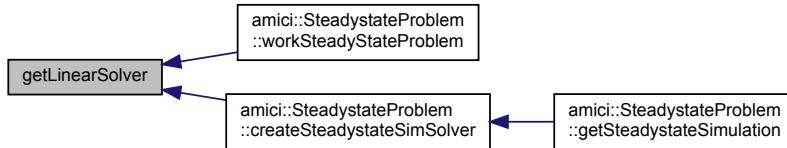
**10.24.3.53 getLinearSolver()**

```
LinearSolver getLinearSolver ( ) const
```

**Returns**

Definition at line 669 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.54 setLinearSolver()

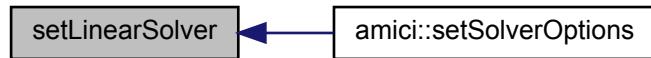
```
void setLinearSolver (
    LinearSolver linsol )
```

##### Parameters

<i>linsol</i>	<input type="text"/>
---------------	----------------------

Definition at line 673 of file solver.cpp.

Here is the caller graph for this function:



#### 10.24.3.55 getInternalSensitivityMethod()

```
InternalSensitivityMethod getInternalSensitivityMethod ( ) const
```

##### Returns

internal sensitivity method

Definition at line 681 of file solver.cpp.

#### 10.24.3.56 setInternalSensitivityMethod()

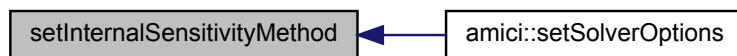
```
void setInternalSensitivityMethod (
    InternalSensitivityMethod ism )
```

**Parameters**

<i>ism</i>	internal sensitivity method
------------	-----------------------------

Definition at line 685 of file solver.cpp.

Here is the caller graph for this function:

**10.24.3.57 init()**

```

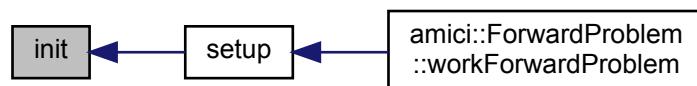
virtual void init (
    AmiVector * x,
    AmiVector * dx,
    realtype t ) [protected], [pure virtual]
  
```

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:

**10.24.3.58 binit()**

```

virtual void binit (
    int which,
  
```

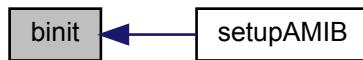
```
AmiVector * xB,
AmiVector * dxB,
realtype t ) [protected], [pure virtual]
```

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:



#### 10.24.3.59 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:



**10.24.3.60 rootInit()**

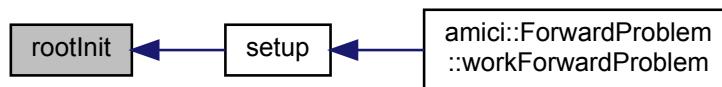
```
virtual void rootInit (
    int ne ) [protected], [pure virtual]
```

RootInit initialises the rootfinding for events

**Parameters**

<i>ne</i>	number of different events
-----------	----------------------------

Here is the caller graph for this function:

**10.24.3.61 sensInit1()**

```
virtual void sensInit1 (
    AmiVectorArray * sx,
    AmiVectorArray * sdx,
    int nplist ) [protected], [pure virtual]
```

SensInit1 initialises the sensitivities at the specified initial timepoint

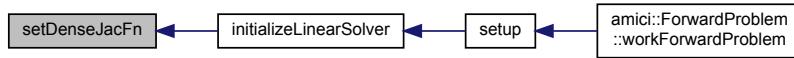
**Parameters**

<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

**10.24.3.62 setDenseJacFn()**

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

SetDenseJacFn sets the dense Jacobian function Here is the caller graph for this function:



#### 10.24.3.63 setSparseJacFn()

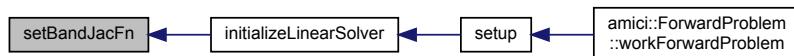
```
virtual void setSparseJacFn ( ) [protected], [pure virtual]
```

SetSparseJacFn sets the sparse Jacobian function

#### 10.24.3.64 setBandJacFn()

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

SetBandJacFn sets the banded Jacobian function Here is the caller graph for this function:



#### 10.24.3.65 setJacTimesVecFn()

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

SetJacTimesVecFn sets the Jacobian vector multiplication function

#### 10.24.3.66 setDenseJacFnB()

```
virtual void setDenseJacFnB (
    int which ) [protected], [pure virtual]
```

SetDenseJacFnB sets the dense Jacobian function

##### Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Here is the caller graph for this function:



#### 10.24.3.67 setSparseJacFnB()

```
virtual void setSparseJacFnB (
    int which ) [protected], [pure virtual]
```

SetSparseJacFn sets the sparse Jacobian function

##### Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

#### 10.24.3.68 setBandJacFnB()

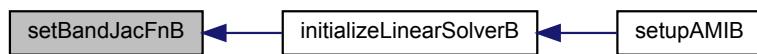
```
virtual void setBandJacFnB (
    int which ) [protected], [pure virtual]
```

SetBandJacFn sets the banded Jacobian function

##### Parameters

<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Here is the caller graph for this function:



#### 10.24.3.69 setJacTimesVecFnB()

```
virtual void setJacTimesVecFnB (
    int which ) [protected], [pure virtual]
```

SetJacTimesVecFn sets the Jacobian vector multiplication function

#### Parameters

<code>which</code>	identifier of the backwards problem
--------------------	-------------------------------------

#### 10.24.3.70 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

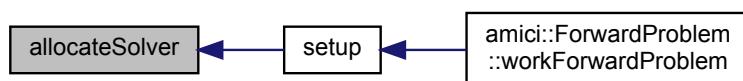
<code>error_code</code>	error identifier
<code>module</code>	name of the module in which the error occurred
<code>function</code>	name of the function in which the error occurred
<b>Type:</b>	char
<code>msg</code>	error message
<code>eh_data</code>	unused input

Definition at line 149 of file solver.cpp.

#### 10.24.3.71 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:



**10.24.3.72 setSStolerances()**

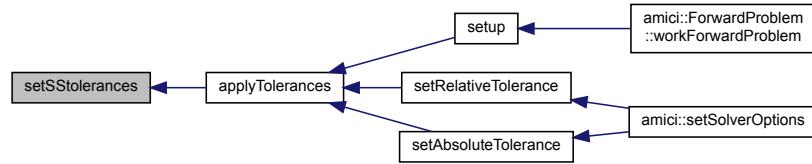
```
virtual void setSStolerances (
    double rtol,
    double atol ) [protected], [pure virtual]
```

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:

**10.24.3.73 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

**10.24.3.74 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
-------------------	-----------------

### 10.24.3.75 setQuadErrConB()

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

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

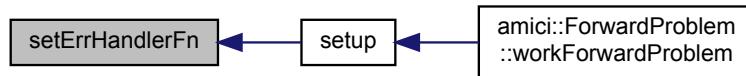
#### Parameters

<i>which</i>	identifier of the backwards problem
<i>flag</i>	activation flag

### 10.24.3.76 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:



### 10.24.3.77 setUserData()

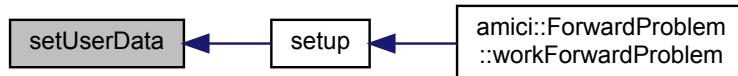
```
virtual void setUserData (
    Model * model ) [protected], [pure virtual]
```

SetUserData attaches the user data instance (here this is a [Model](#)) to the forward problem

#### Parameters

<i>model</i>	<a href="#">Model</a> instance,
--------------	---------------------------------

Here is the caller graph for this function:



#### 10.24.3.78 setUserDataB()

```

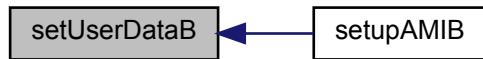
virtual void setUserDataB (
    int which,
    Model * model ) [protected], [pure virtual]
  
```

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>	<a href="#">Model</a> instance,

Here is the caller graph for this function:



#### 10.24.3.79 setMaxNumSteps()

```

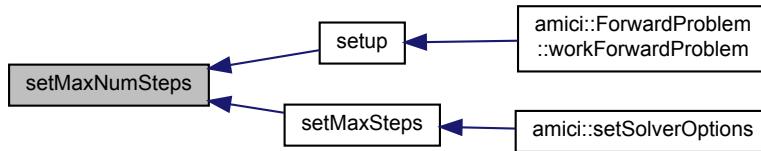
virtual void setMaxNumSteps (
    long int mxsteps ) [protected], [pure virtual]
  
```

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:



#### 10.24.3.80 setMaxNumStepsB()

```

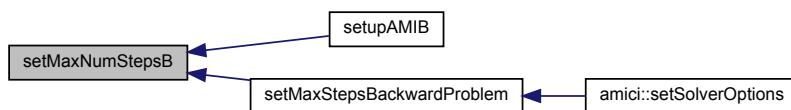
virtual void setMaxNumStepsB (
    int which,
    long int mxstepsB ) [protected], [pure virtual]
  
```

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:



#### 10.24.3.81 setStabLimDet()

```

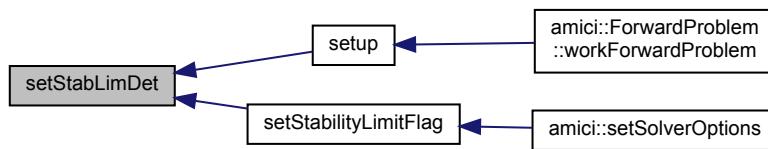
virtual void setStabLimDet (
    int stldet ) [protected], [pure virtual]
  
```

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:



#### 10.24.3.82 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:



#### 10.24.3.83 setId()

```

virtual void setId (
    Model * model ) [protected], [pure virtual]
  
```

setId specify algebraic/differential components (DAE only)

##### Parameters

<i>model</i>	model specification
--------------	---------------------

#### 10.24.3.84 setSuppressAlg()

```
virtual void setSuppressAlg (
    bool flag ) [protected], [pure virtual]
```

SetId deactivates error control for algebraic components (DAE only)

##### Parameters

<i>flag</i>	deactivation flag
-------------	-------------------

#### 10.24.3.85 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

#### 10.24.3.86 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

#### 10.24.3.87 adjInit()

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

AdjInit initializes the adjoint problem

10.24.3.88 `allocateSolverB()`

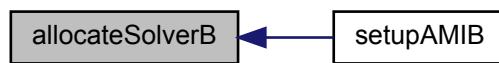
```
virtual void allocateSolverB (
    int * which ) [protected], [pure virtual]
```

specifies solver method and initializes solver memory for the backward problem

## Parameters

<code>which</code>	identifier of the backwards problem
--------------------	-------------------------------------

Here is the caller graph for this function:

10.24.3.89 `setSStolerancesB()`

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

SStolerancesB sets relative and absolute tolerances for the backward problem

## Parameters

<code>which</code>	identifier of the backwards problem
<code>relTolB</code>	relative tolerances
<code>absTolB</code>	absolute tolerances

10.24.3.90 `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

**10.24.3.91 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:

**10.24.3.92 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:



#### 10.24.3.93 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:



#### 10.24.3.94 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:



#### 10.24.3.95 diag()

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

Diag attaches a diagonal linear solver to the forward problem

#### 10.24.3.96 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
--------------	-------------------------------------

#### 10.24.3.97 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

#### 10.24.3.98 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

#### 10.24.3.99 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

#### 10.24.3.100 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

#### 10.24.3.101 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

**10.24.3.102 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

**10.24.3.103 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

**10.24.3.104 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
-----------------	--

**10.24.3.105 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

**10.24.3.106 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

**10.24.3.107 getNumSteps()**

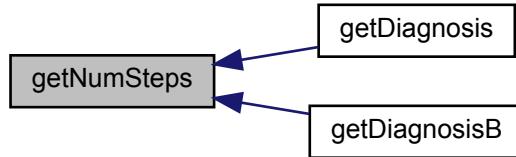
```
virtual void getNumSteps (
    void * ami_mem,
    long int * numsteps) const [protected], [pure virtual]
```

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:



#### 10.24.3.108 `getNumRhsEvals()`

```

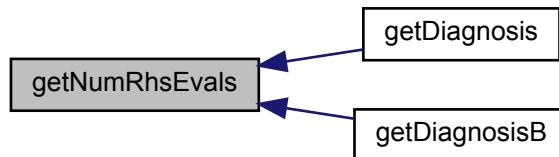
virtual void getNumRhsEvals (
    void * ami_mem,
    long int * numrhsevals ) const [protected], [pure virtual]
  
```

`getNumRhsEvals` reports the number of right hand evaluations

##### Parameters

<code>ami_mem</code>	pointer to the solver memory instance (can be from forward or backward problem)
<code>numrhsevals</code>	output array

Here is the caller graph for this function:



#### 10.24.3.109 `getNumErrTestFails()`

```

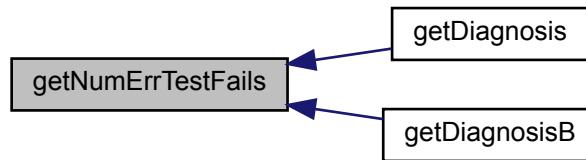
virtual void getNumErrTestFails (
    void * ami_mem,
    long int * numerertestfails ) const [protected], [pure virtual]
  
```

`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:

**10.24.3.110 getNumNonlinSolvConvFails()**

```

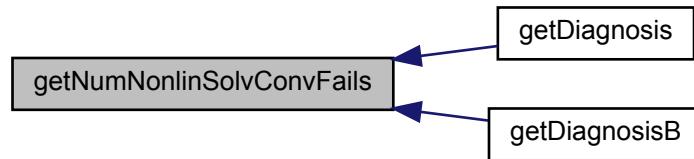
virtual void getNumNonlinSolvConvFails (
    void * ami_mem,
    long int * numnonlinsolvconvfails ) const [protected], [pure virtual]
  
```

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:



**10.24.3.111 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:

**10.24.3.112 initializeLinearSolver()**

```
void initializeLinearSolver (
    const Model * model ) [protected]
```

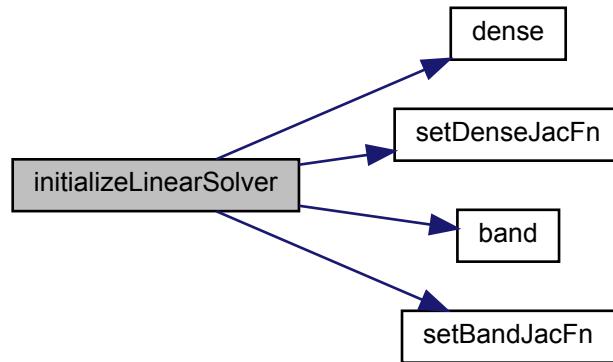
initializeLinearSolver sets the linear solver for the forward problem

**Parameters**

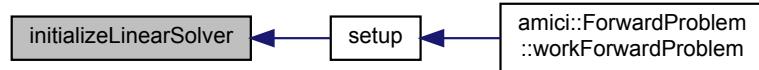
<i>model</i>	pointer to the model object
--------------	-----------------------------

Definition at line 227 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.24.3.113 initializeLinearSolverB()

```
void initializeLinearSolverB (
    const Model * model,
    const int which ) [protected]
```

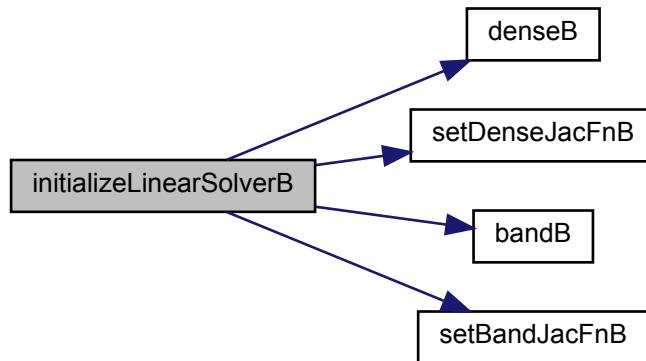
Sets the linear solver for the backward problem

##### 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:



#### 10.24.3.114 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

#### 10.24.3.115 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

**10.24.3.116 getModel()**

```
virtual const Model* getModel ( ) const [protected], [pure virtual]
```

Accessor function to the model stored in the user data

**Returns**

user data model

**10.24.3.117 getMallocDone()**

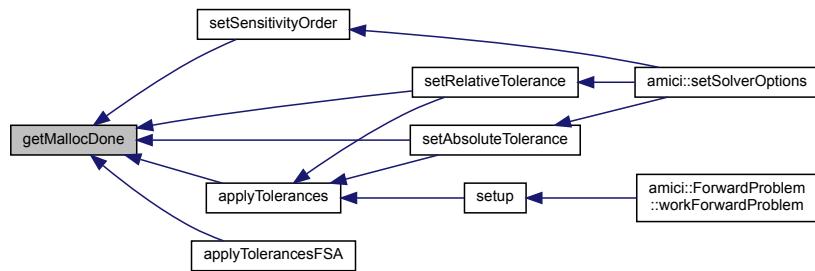
```
virtual bool getMallocDone ( ) const [protected], [pure virtual]
```

checks whether memory for the forward problem has been allocated

**Returns**

solverMemory->(cv|ida)\_\_\_MallocDone

Here is the caller graph for this function:

**10.24.3.118 getAdjMallocDone()**

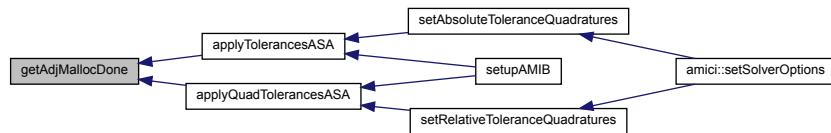
```
virtual bool getAdjMallocDone ( ) const [protected], [pure virtual]
```

checks whether memory for the backward problem has been allocated

**Returns**

solverMemory->(cv|ida)\_\_\_adjMallocDone

Here is the caller graph for this function:



### 10.24.3.119 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\_mem* pointer to the backward solver memory instance

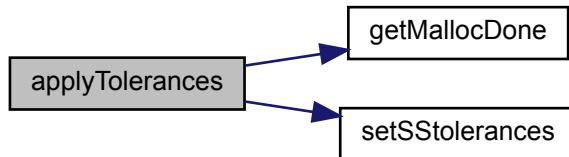
### 10.24.3.120 applyTolerances()

```
void applyTolerances ( ) [protected]
```

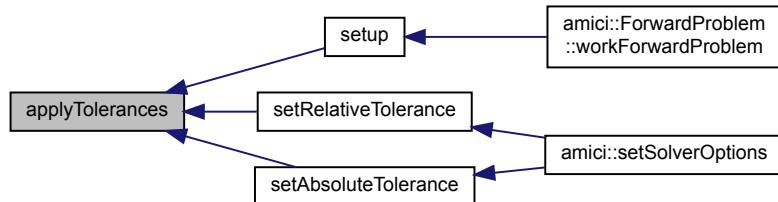
updates solver tolerances according to the currently specified member variables

Definition at line 403 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



**10.24.3.121 applyTolerancesFSA()**

```
void applyTolerancesFSA ( ) [protected]
```

updates FSA solver tolerances according to the currently specified member variables

Definition at line 410 of file solver.cpp.

Here is the call graph for this function:

**10.24.3.122 applyTolerancesASA()**

```
void applyTolerancesASA ( int which ) [protected]
```

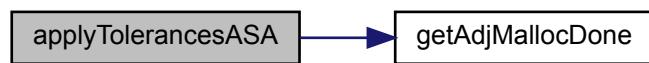
updates ASA solver tolerances according to the currently specified member variables

**Parameters**

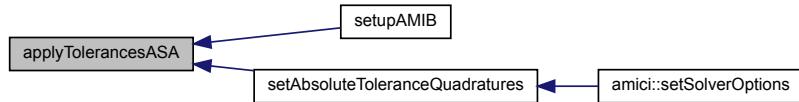
<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Definition at line 424 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.24.3.123 applyQuadTolerancesASA()

```
void applyQuadTolerancesASA (
    int which ) [protected]
```

updates ASA quadrature solver tolerances according to the currently specified member variables

##### Parameters

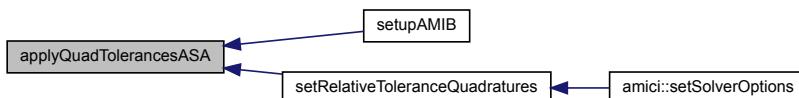
<i>which</i>	identifier of the backwards problem
--------------	-------------------------------------

Definition at line 435 of file solver.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



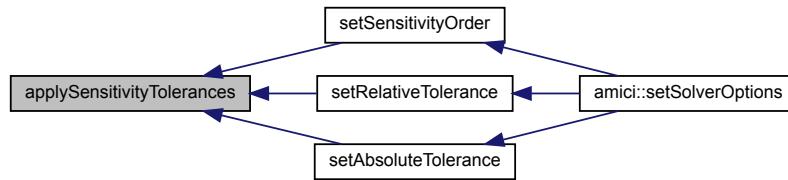
## 10.24.3.124 applySensitivityTolerances()

```
void applySensitivityTolerances ( ) [protected]
```

updates all sensitivity solver tolerances according to the currently specified member variables

Definition at line 454 of file solver.cpp.

Here is the caller graph for this function:



## 10.24.4 Friends And Related Function Documentation

## 10.24.4.1 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

## 10.24.4.2 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.

**10.24.5 Member Data Documentation****10.24.5.1 solverMemory**

```
std::unique_ptr<void, std::function<void(void *)> > solverMemory [protected]
```

pointer to solver memory block

Definition at line 1030 of file solver.h.

**10.24.5.2 solverMemoryB**

```
std::vector<std::unique_ptr<void, std::function<void(void *)> > > solverMemoryB [protected]
```

pointer to solver memory block

Definition at line 1033 of file solver.h.

**10.24.5.3 solverWasCalled**

```
bool solverWasCalled = false [protected]
```

flag indicating whether the solver was called

Definition at line 1036 of file solver.h.

**10.24.5.4 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 1040 of file solver.h.

#### 10.24.5.5 lmm

```
LinearMultistepMethod lmm = LinearMultistepMethod::BDF [protected]
```

specifies the linear multistep method.

Definition at line 1044 of file solver.h.

#### 10.24.5.6 iter

```
NonlinearSolverIteration iter = NonlinearSolverIteration::newton [protected]
```

specifies the type of nonlinear solver iteration

Definition at line 1049 of file solver.h.

#### 10.24.5.7 interpType

```
InterpolationType interpType = InterpolationType::hermite [protected]
```

interpolation type for the forward problem solution which is then used for the backwards problem.

Definition at line 1054 of file solver.h.

#### 10.24.5.8 maxsteps

```
int maxsteps = 10000 [protected]
```

maximum number of allowed integration steps

Definition at line 1057 of file solver.h.

## 10.25 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)
- void [applyNewtonsMethod](#) (ReturnData \*rdata, Model \*model, NewtonSolver \*newtonSolver, int newton\_try)
- void [getNewtonOutput](#) (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)

### 10.25.1 Detailed Description

Definition at line 25 of file steadystateproblem.h.

### 10.25.2 Constructor & Destructor Documentation

#### 10.25.2.1 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 50 of file steadystateproblem.h.

### 10.25.3 Member Function Documentation

#### 10.25.3.1 workSteadyStateProblem()

```
void workSteadyStateProblem (
    ReturnData * rdata,
    Solver * solver,
    Model * model,
    int it )
```

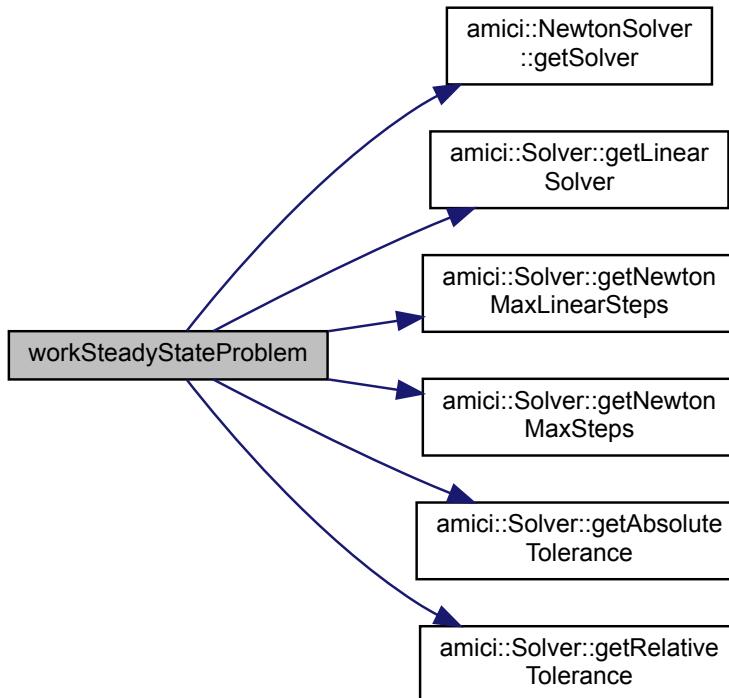
Tries to determine the steady state of the ODE system by a Newton solver, uses forward intergration, if the Newton solver fails, restarts Newton solver, if integration fails. Computes steady state sensitivities

#### 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:



### 10.25.3.2 applyNewtonsMethod()

```

void applyNewtonsMethod (
    ReturnData * rdata,
    Model * model,
    NewtonSolver * newtonSolver,
    int newton_try )
  
```

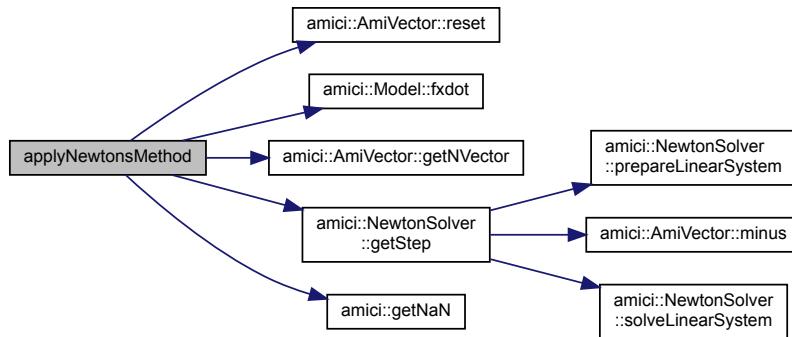
`applyNewtonsMethod` applies Newtons method to the current state  $x$  to find the steady state Runs the Newton solver iterations and checks for convergence to steady state

#### Parameters

<code>rdata</code>	pointer to the return data object
<code>model</code>	pointer to the AMICI model object
<code>newtonSolver</code>	pointer to the <code>NewtonSolver</code> object <b>Type:</b> <code>NewtonSolver</code>
<code>newton_try</code>	integer start number of Newton solver (1 or 2)

Definition at line 92 of file `steadystateproblem.cpp`.

Here is the call graph for this function:



### 10.25.3.3 getNewtonOutput()

```

void getNewtonOutput (
    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 200 of file steadystateproblem.cpp.

Here is the call graph for this function:



## 10.25.3.4 getSteadystateSimulation()

```
void getSteadystateSimulation (
    ReturnData * rdata,
    Solver * solver,
    Model * model,
    int it )
```

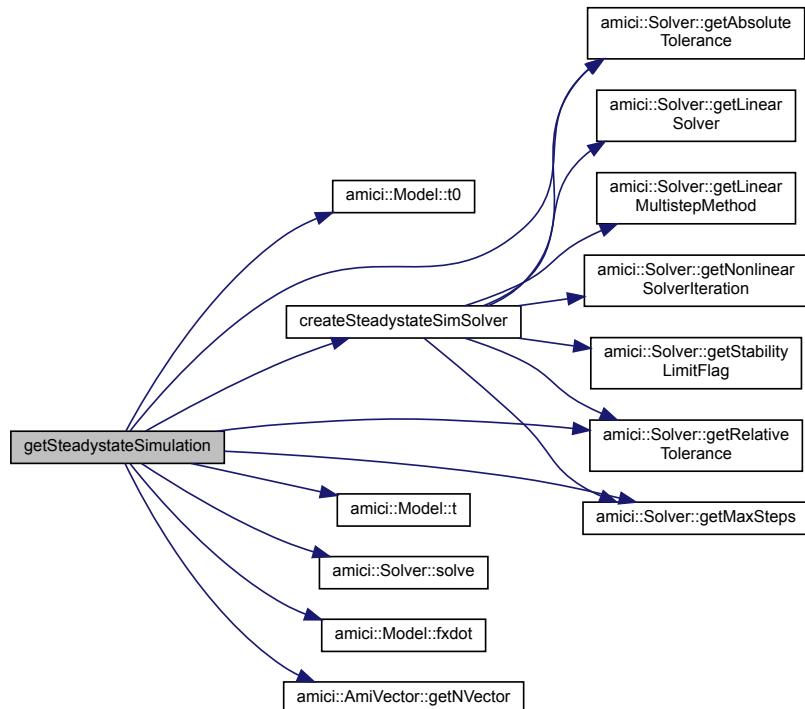
Forward simulation is launched, if Newton solver fails in first try

## 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 233 of file steadystateproblem.cpp.

Here is the call graph for this function:



## 10.25.3.5 createSteadystateSimSolver()

```
std::unique_ptr< CVodeSolver > createSteadystateSimSolver (
    Solver * solver,
```

```
Model * model,
realtype tstart )
```

initialize CVodeSolver instance for preequilibration simulation

#### Parameters

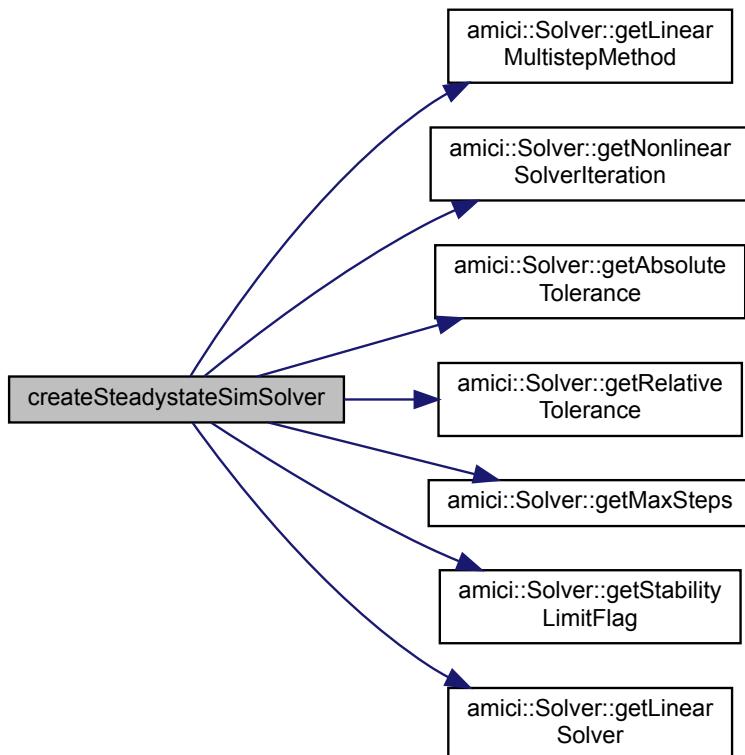
<i>solver</i>	pointer to the AMICI solver object
<i>model</i>	pointer to the AMICI model object
<i>tstart</i>	time point for starting Newton simulation

#### Returns

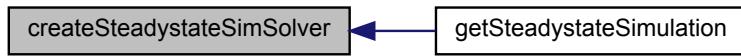
solver instance

Definition at line 300 of file steadystateproblem.cpp.

Here is the call graph for this function:



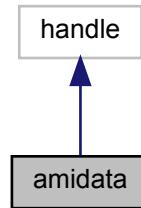
Here is the caller graph for this function:



## 10.26 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("")

- matlabypesubstitute **Y** = double.empty("")  
*timepoints of observations*
- matlabypesubstitute **Sigma\_Y** = double.empty("")  
*observations*
- matlabypesubstitute **Sigma\_Z** = double.empty("")  
*standard deviation of observations*
- matlabypesubstitute **Z** = double.empty("")  
*event observations*
- matlabypesubstitute **Sigma\_Z** = double.empty("")  
*standard deviation of event observations*
- matlabypesubstitute **condition** = double.empty("")  
*experimental condition*
- matlabypesubstitute **conditionPreequilibration** = double.empty("")  
*experimental condition for preequilibration*

### 10.26.1 Detailed Description

Definition at line 17 of file amidata.m.

### 10.26.2 Constructor & Destructor Documentation

#### 10.26.2.1 amidata()

```
amidata (
    matlabypesubstitute 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

<code>varargin</code>	<input type="button" value=""/>
-----------------------	---------------------------------

Definition at line 130 of file amidata.m.

### 10.26.3 Member Data Documentation

**10.26.3.1 nt**

```
nt = 0
```

**Default:** 0**Note**

This property has custom functionality when its value is changed.

Definition at line 31 of file amidata.m.

**10.26.3.2 ny**

```
ny = 0
```

**Default:** 0**Note**

This property has custom functionality when its value is changed.

Definition at line 39 of file amidata.m.

**10.26.3.3 nz**

```
nz = 0
```

**Default:** 0**Note**

This property has custom functionality when its value is changed.

Definition at line 47 of file amidata.m.

**10.26.3.4 ne**

```
ne = 0
```

**Default:** 0**Note**

This property has custom functionality when its value is changed.

Definition at line 55 of file amidata.m.

### 10.26.3.5 nk

```
nk = 0
```

#### **Default:** 0

##### Note

This property has custom functionality when its value is changed.

Definition at line 63 of file amidata.m.

### 10.26.3.6 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.

### 10.26.3.7 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.

### 10.26.3.8 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.

### 10.26.3.9 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.

### 10.26.3.10 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.

### 10.26.3.11 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.

### 10.26.3.12 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.

## 10.27 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.*
- **mlhsInnnerSubst< 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*

### 10.27.1 Detailed Description

Definition at line 17 of file amievent.m.

### 10.27.2 Constructor & Destructor Documentation

#### 10.27.2.1 amievent()

```
amievent (
    matlabtypesubstitute trigger,
    matlabtypesubstitute bolus,
    matlabtypesubstitute z )
```

#### Parameters

<b>trigger</b>	trigger function, the event will be triggered on at all roots of this function
<b>bolus</b>	the bolus that will be added to all states on every occurrence of the event
<b>z</b>	the event output that will be reported on every occurrence of the event

Definition at line 75 of file amievent.m.

## 10.27.3 Member Function Documentation

## 10.27.3.1 setHflag()

```
mlhsInnerSubst<::amievent> setHflag( ::double hflag )
```

## Parameters

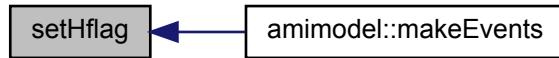
<i>hflag</i>	value for the hflag property
--------------	------------------------------

## 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:



## 10.27.4 Member Data Documentation

## 10.27.4.1 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.

#### 10.27.4.2 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.

#### 10.27.4.3 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.

#### 10.27.4.4 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.

### 10.28 amifun Class Reference

AMIFUN defines functions which later on will be transformed into appropriate C code.

### Public Member Functions

- **amifun** (matlabtypesubstitute **funstr**, matlabtypesubstitute **model**)
 

*amevent 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*
- matlabtypesubstitute **nvecs** = cell.empty("")
 

*nvec dependencies*
- matlabtypesubstitute **sensiflag** = logical.empty("")
 

*indicates whether the function is a sensitivity or derivative with respect to parameters*

### 10.28.1 Detailed Description

Definition at line 17 of file amifun.m.

### 10.28.2 Constructor & Destructor Documentation

#### 10.28.2.1 amifun()

```
amifun (
    matlabtypesubstitute funstr,
    matlabtypesubstitute 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.

### 10.28.3 Member Function Documentation

#### 10.28.3.1 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.

#### 10.28.3.2 writeCcode()

```
noret::substitute writeCcode (
    ::amimodel model,
    ::fileid fid )
```

**Parameters**

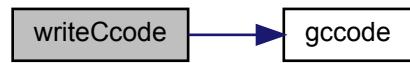
<i>model</i>	model defintion object
<i>fid</i>	file id in which the final expression is written

**Return values**

<i>fid</i>	void
------------	------

Definition at line 18 of file writeCcode.m.

Here is the call graph for this function:

**10.28.3.3 writeMcode()**

```
noret::substitute writeMcode ( ::amimodel model )
```

**Parameters**

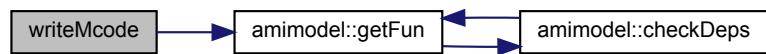
<i>model</i>	model defintion object
--------------	------------------------

**Return values**

<i>model</i>	void
--------------	------

Definition at line 18 of file writeMcode.m.

Here is the call graph for this function:



#### 10.28.3.4 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:



#### 10.28.3.5 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.

#### 10.28.3.6 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.

**10.28.3.7 getNVecs()**

```
mlhsInnerSubst<::amifun > getNVecs ( )
```

**Return values**

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getNVecs.m.

**10.28.3.8 getCVar()**

```
mlhsInnerSubst<::amifun > getCVar ( )
```

**Return values**

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getCVar.m.

**10.28.3.9 getSensiFlag()**

```
mlhsInnerSubst<::amifun > getSensiFlag ( )
```

**Return values**

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getSensiFlag.m.

**10.28.3.10 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.

Here is the call graph for this function:



## 10.28.4 Member Data Documentation

### 10.28.4.1 sym

```
sym = sym(" [ ]")
```

**Default:** sym("[ ]")

Definition at line 27 of file amifun.m.

### 10.28.4.2 sym\_noopt

```
sym_noopt = sym(" [ ]")
```

**Default:** sym("[ ]")

Definition at line 35 of file amifun.m.

#### 10.28.4.3 strsym

```
strsym = sym("[ ]")
```

**Default:** `sym("[ ])")`

Definition at line 43 of file amifun.m.

#### 10.28.4.4 strsym\_old

```
strsym_old = sym("[ ]")
```

**Default:** `sym("[ ])")`

Definition at line 51 of file amifun.m.

#### 10.28.4.5 funstr

```
funstr = char.empty("")
```

**Default:** `char.empty("")`

Definition at line 59 of file amifun.m.

#### 10.28.4.6 cvar

```
cvar = char.empty("")
```

**Default:** `char.empty("")`

Definition at line 67 of file amifun.m.

#### 10.28.4.7 argstr

```
argstr = char.empty("")
```

**Default:** `char.empty("")`

Definition at line 75 of file amifun.m.

#### 10.28.4.8 deps

```
deps = cell.empty("")
```

**Default:** cell.empty("")

Definition at line 83 of file amifun.m.

#### 10.28.4.9 nvecs

```
nvecs = cell.empty("")
```

**Default:** cell.empty("")

Definition at line 91 of file amifun.m.

#### 10.28.4.10 sensiflag

```
sensiflag = logical.empty("")
```

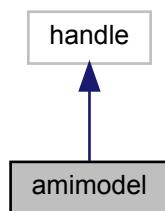
**Default:** logical.empty("")

Definition at line 99 of file amifun.m.

### 10.29 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*
- mlhsInnnerSubst< matlabtypesubstitute > **checkDeps** (::struct **HTable**,::cell **deps**)
 

*checkDeps checks the dependencies of functions and populates sym fields if necessary*
- mlhsInnnerSubst< matlabtypesubstitute > **loadOldHashes** ()
 

*loadOldHashes loads information from a previous compilation of the model.*
- mlhsInnnerSubst< matlabtypesubstitute > **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.*
- mlhsInnnerSubst< matlabtypesubstitute > **augmento2vec** ()
 

*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.*

## Static Public Member Functions

- static noret::substitute **compileAndLinkModel** (matlabtypesubstitute **modelName**, matlabtypesubstitute **modelSourceFolder**, matlabtypesubstitute **coptim**, matlabtypesubstitute **debug**, matlabtypesubstitute **funs**, matlabtypesubstitute **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** (matlabtypesubstitute **nx**, matlabtypesubstitute **ny**, matlabtypesubstitute **np**, matlabtypesubstitute **nk**, matlabtypesubstitute **nz**, matlabtypesubstitute **o2flag**, ::amimodel **amimodelo2**, matlabtypesubstitute **wrapperFilename**, matlabtypesubstitute **modelName**, matlabtypesubstitute **pscale**, matlabtypesubstitute **forward**, matlabtypesubstitute **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 **ntrue** = 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 **nnz** = 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("")  
*cell array of matlab functions to be compiled*
- ::string **coptim** = "-O3"  
*optimisation flag for compilation*
- ::string **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*

### 10.29.1 Detailed Description

Definition at line 17 of file amimodel.m.

## 10.29.2 Constructor & Destructor Documentation

### 10.29.2.1 amimodel()

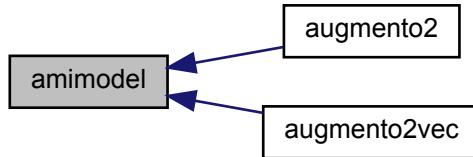
```
amimodel (
    ::string symfun,
    ::string modelname )
```

#### Parameters

<i>symfun</i>	this is the string to the function which generates the modelstruct. You can also directly pass the struct here
<i>modelname</i>	name of the model

Definition at line 515 of file amimodel.m.

Here is the caller graph for this function:



## 10.29.3 Member Function Documentation

### 10.29.3.1 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.

Here is the caller graph for this function:



#### 10.29.3.2 update modelName()

```
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.

#### 10.29.3.3 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.

#### 10.29.3.4 parseModel()

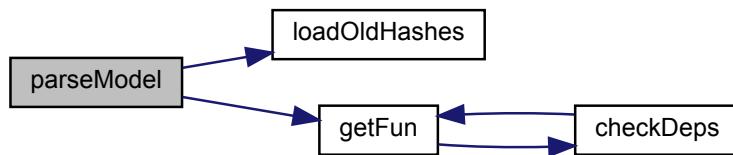
```
noret::substitute parseModel ( )
```

**Return values**

void	
------	--

Definition at line 18 of file parseModel.m.

Here is the call graph for this function:

**10.29.3.5 generateC()**

```
noret::substitute generateC ( )
```

**Return values**

void	
------	--

Definition at line 18 of file generateC.m.

**10.29.3.6 compileC()**

```
noret::substitute compileC ( )
```

**Return values**

void	
------	--

Definition at line 18 of file compileC.m.

Here is the call graph for this function:



#### 10.29.3.7 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>	void
-------------------	------

Definition at line 18 of file generateM.m.

Here is the call graph for this function:



#### 10.29.3.8 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**

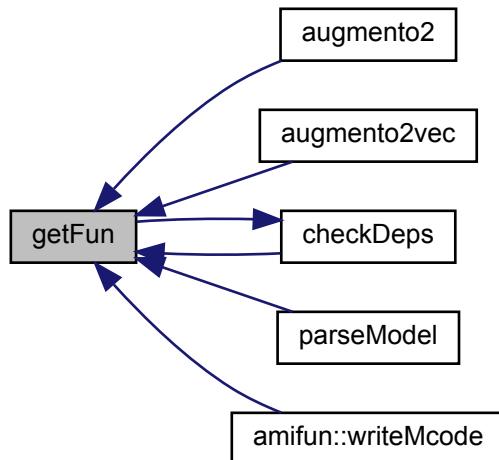
<i>funstr</i>	void
---------------	------

Definition at line 18 of file getFun.m.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.29.3.9 makeEvents()**

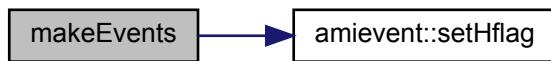
```
noret::substitute makeEvents ( )
```

**Return values**

<i>void</i>	
-------------	--

Definition at line 18 of file makeEvents.m.

Here is the call graph for this function:



#### 10.29.3.10 makeSym()

`noret::substitute makeSym ( )`

##### Return values

<code>void</code>	
-------------------	--

Definition at line 18 of file `makeSym.m`.

#### 10.29.3.11 checkDeps()

```
mlhsInnerSubst<::bool > checkDeps (
    ::struct HTable,
    ::cell deps )
```

##### Parameters

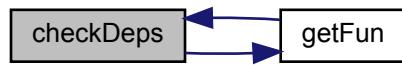
<code>HTable</code>	struct with reference hashes of functions in its fields
<code>deps</code>	cell array with containing a list of dependencies

##### Return values

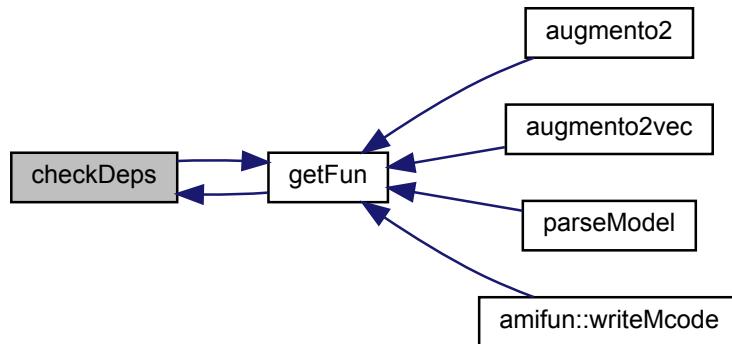
<code>cflag</code>	boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable
--------------------	---

Definition at line 18 of file `checkDeps.m`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 10.29.3.12 loadOldHashes()

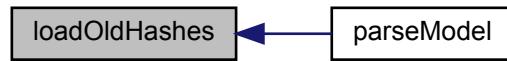
```
mlhsInnerSubst<::struct> loadOldHashes( )
```

#### 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:



#### 10.29.3.13 augmento2()

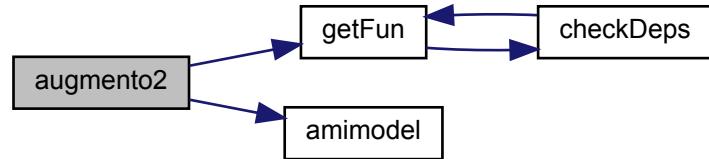
`mlhsInnerSubst< matlabtypesubstitute > augmento2 ( )`

##### 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:



#### 10.29.3.14 augmento2vec()

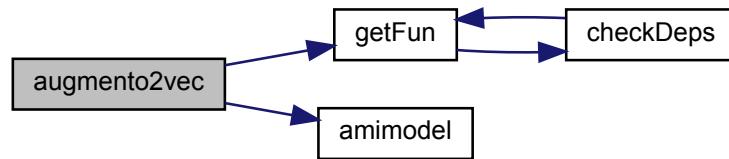
`mlhsInnerSubst<::amimodel > augmento2vec ( )`

##### 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:



#### 10.29.3.15 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 <a href="#">amiwrap()</a>
<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:



## 10.29.3.16 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:



## 10.29.4 Member Data Documentation

#### 10.29.4.1 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.

#### 10.29.4.2 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.

#### 10.29.4.3 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.

#### 10.29.4.4 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.

#### 10.29.4.5 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.

#### 10.29.4.6 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.

#### 10.29.4.7 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.

#### 10.29.4.8 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.

#### 10.29.4.9 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.

#### 10.29.4.10 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.

#### 10.29.4.11 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.

#### 10.29.4.12 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.

**10.29.4.13 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.

**10.29.4.14 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.

**10.29.4.15 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.

**10.29.4.16 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.

#### 10.29.4.17 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.

#### 10.29.4.18 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.

#### 10.29.4.19 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.

#### 10.29.4.20 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.

**10.29.4.21 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.

**10.29.4.22 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.

**10.29.4.23 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.

**10.29.4.24 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.

#### 10.29.4.25 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.

#### 10.29.4.26 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.

#### 10.29.4.27 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.

#### 10.29.4.28 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.

**10.29.4.29 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.

**10.29.4.30 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.

**10.29.4.31 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.

**10.29.4.32 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.

#### 10.29.4.33 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.

#### 10.29.4.34 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.

#### 10.29.4.35 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.

#### 10.29.4.36 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.

#### 10.29.4.37 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.

#### 10.29.4.38 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.

#### 10.29.4.39 z2event

```
z2event = double.empty("")
```

**Default:** double.empty("")

Definition at line 455 of file amimodel.m.

#### 10.29.4.40 splineflag

```
splineflag = false
```

**Default:** false

Definition at line 463 of file amimodel.m.

**10.29.4.41 minflag**

```
minflag = false
```

**Default:** false

Definition at line 471 of file amimodel.m.

**10.29.4.42 maxflag**

```
maxflag = false
```

**Default:** false

Definition at line 479 of file amimodel.m.

**10.29.4.43 nw**

```
nw = 0
```

**Default:** 0

Definition at line 487 of file amimodel.m.

**10.29.4.44 ndwdx**

```
ndwdx = 0
```

**Default:** 0

Definition at line 496 of file amimodel.m.

**10.29.4.45 ndwdp**

```
ndwdp = 0
```

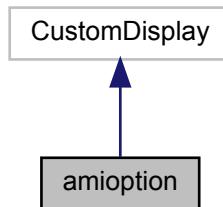
**Default:** 0

Definition at line 504 of file amimodel.m.

## 10.30 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 integration tolerance*
- matlabtypesubstitute **quad\_rtol** = 1e-8  
*relative integration tolerance*
- matlabtypesubstitute **maxstepsB** = 0  
*maximum number of integration steps*
- matlabtypesubstitute **sens\_ind** = double.empty("")  
*index of parameters for which the sensitivities are computed*
- matlabtypesubstitute **tstart** = 0  
*starting time of the simulation*
- matlabtypesubstitute **lmm** = 2  
*linear multistep method.*
- matlabtypesubstitute **iter** = 2  
*iteration method for linear multistep.*
- matlabtypesubstitute **linsol** = 9  
*linear solver*
- matlabtypesubstitute **stldet** = true

*stability detection flag*

- matlabypesubstitute `interpType` = 1  
*interpolation type*
- matlabypesubstitute `ism` = 1  
*forward sensitivity mode*
- matlabypesubstitute `sensi_meth` = 1  
*sensitivity method*
- matlabypesubstitute `sensi` = 0  
*sensitivity order*
- matlabypesubstitute `nmaxevent` = 10  
*number of reported events*
- matlabypesubstitute `ordering` = 0  
*reordering of states*
- matlabypesubstitute `ss` = 0  
*steady state sensitivity flag*
- matlabypesubstitute `x0` = double.empty("")  
*custom initial state*
- matlabypesubstitute `sx0` = double.empty("")  
*custom initial sensitivity*
- matlabypesubstitute `newton_maxsteps` = 40  
*newton solver: maximum newton steps*
- matlabypesubstitute `newton_maxlinsteps` = 100  
*newton solver: maximum linear steps*
- matlabypesubstitute `newton_preq` = false  
*preequilibration of system via newton solver*
- matlabypesubstitute `z2event` = double.empty("")  
*mapping of event outputs to events*
- matlabypesubstitute `pscale` = "[ ]"  
*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).*

### 10.30.1 Detailed Description

Definition at line 17 of file amioption.m.

### 10.30.2 Constructor & Destructor Documentation

#### 10.30.2.1 amioption()

```
amioption (
    matlabypesubstitute 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**

<i>varargin</i>	input to construct amioption object, see function function description
-----------------	--

Definition at line 243 of file amioption.m.

### 10.30.3 Member Data Documentation

#### 10.30.3.1 atol

```
atol = 1e-16
```

**Default:** 1e-16

Definition at line 28 of file amioption.m.

#### 10.30.3.2 rtol

```
rtol = 1e-8
```

**Default:** 1e-8

Definition at line 36 of file amioption.m.

#### 10.30.3.3 maxsteps

```
maxsteps = 1e4
```

**Default:** 1e4

Definition at line 44 of file amioption.m.

#### 10.30.3.4 quad\_atol

```
quad_atol = 1e-12
```

**Default:** 1e-12

Definition at line 52 of file amioption.m.

### 10.30.3.5 quad\_rtol

```
quad_rtol = 1e-8
```

**Default:** 1e-8

Definition at line 60 of file amioption.m.

### 10.30.3.6 maxstepsB

```
maxstepsB = 0
```

**Default:** 0

Definition at line 68 of file amioption.m.

### 10.30.3.7 sens\_ind

```
sens_ind = double.empty("")
```

**Default:** double.empty("")

Definition at line 76 of file amioption.m.

### 10.30.3.8 tstart

```
tstart = 0
```

**Default:** 0

Definition at line 84 of file amioption.m.

### 10.30.3.9 lmm

```
lmm = 2
```

**Default:** 2

Definition at line 92 of file amioption.m.

**10.30.3.10 iter**

```
iter = 2
```

**Default:** 2

Definition at line 100 of file amioption.m.

**10.30.3.11 linsol**

```
linsol = 9
```

**Default:** 9

Definition at line 108 of file amioption.m.

**10.30.3.12 stldet**

```
stldet = true
```

**Default:** true

Definition at line 116 of file amioption.m.

**10.30.3.13 interpType**

```
interpType = 1
```

**Default:** 1

Definition at line 124 of file amioption.m.

**10.30.3.14 ism**

```
ism = 1
```

**Default:** 1

Definition at line 132 of file amioption.m.

### 10.30.3.15 sensi\_meth

```
sensi_meth = 1
```

**Default:** 1

**Note**

This property has custom functionality when its value is changed.

Definition at line 140 of file amioption.m.

### 10.30.3.16 sensi

```
sensi = 0
```

**Default:** 0

**Note**

This property has custom functionality when its value is changed.

Definition at line 148 of file amioption.m.

### 10.30.3.17 nmaxevent

```
nmaxevent = 10
```

**Default:** 10

Definition at line 156 of file amioption.m.

### 10.30.3.18 ordering

```
ordering = 0
```

**Default:** 0

Definition at line 164 of file amioption.m.

### 10.30.3.19 ss

```
ss = 0
```

**Default:** 0

Definition at line 172 of file amioption.m.

### 10.30.3.20 x0

```
x0 = double.empty("")
```

**Default:** double.empty("")

Definition at line 180 of file amioption.m.

### 10.30.3.21 sx0

```
sx0 = double.empty("")
```

**Default:** double.empty("")

Definition at line 188 of file amioption.m.

### 10.30.3.22 newton\_maxsteps

```
newton_maxsteps = 40
```

**Default:** 40

#### Note

This property has custom functionality when its value is changed.

Definition at line 196 of file amioption.m.

**10.30.3.23 newton\_maxlinsteps**

```
newton_maxlinsteps = 100
```

**Default:** 100**Note**

This property has custom functionality when its value is changed.

Definition at line 204 of file amioption.m.

**10.30.3.24 newton\_preq**

```
newton_preq = false
```

**Default:** false**Note**

This property has custom functionality when its value is changed.

Definition at line 212 of file amioption.m.

**10.30.3.25 z2event**

```
z2event = double.empty("")
```

**Default:** double.empty("")

Definition at line 220 of file amioption.m.

**10.30.3.26 pscale**

```
pscale = "[ ]"
```

**Default:** "[ ]"**Note**

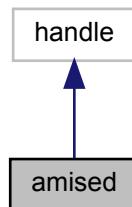
This property has custom functionality when its value is changed.

Definition at line 228 of file amioption.m.

## 10.31 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

- matlabtypesubstitute [model](#) = struct("event",[],'sym',[])  
*amimodel from the specified model*
- matlabtypesubstitute [modelname](#) = {"["](#)}  
*cell array of model identifiers*
- matlabtypesubstitute [sedml](#) = struct.empty("")  
*stores the struct tree from the xml definition*
- matlabtypesubstitute [outputcount](#) = "[ ]"  
*count the number of outputs per model*
- matlabtypesubstitute [varidx](#) = "[ ]"  
*indexes for dataGenerators*
- matlabtypesubstitute [varsym](#) = sym("[ ]")  
*symbolic expressions for variables*
- matlabtypesubstitute [datasym](#) = sym("[ ]")  
*symbolic expressions for data*

#### 10.31.1 Detailed Description

Definition at line 17 of file amised.m.

#### 10.31.2 Constructor & Destructor Documentation

##### 10.31.2.1 amised()

```

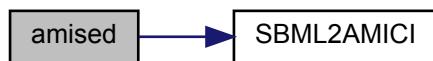
amised (
    matlabtypesubstitute sedname )
  
```

**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:



### 10.31.3 Member Data Documentation

#### 10.31.3.1 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.

#### 10.31.3.2 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.

### 10.31.3.3 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.

### 10.31.3.4 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.

### 10.31.3.5 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.

### 10.31.3.6 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.

### 10.31.3.7 datasym

```
datasym = sym(" [ ]")
```

#### Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

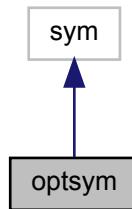
**Default:** sym("[ ]")

Definition at line 93 of file amised.m.

## 10.32 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*

### 10.32.1 Detailed Description

Definition at line 17 of file optsym.m.

### 10.32.2 Constructor & Destructor Documentation

#### 10.32.2.1 `optsym()`

```
optsym (
    ::sym symbol )
```

**Parameters**

<i>symbol</i>	symbolic object
---------------	-----------------

Definition at line 32 of file optsym.m.

**10.32.3 Member Function Documentation****10.32.3.1 getoptimized()**

```
mlhsInnerSubst<::sym > getoptimized ( )
```

**Return values**

<i>out</i>	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**

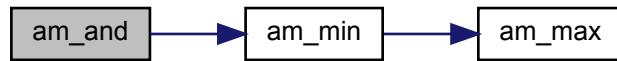
<i>a</i>	first input parameter
<i>b</i>	second input parameter

**Return values**

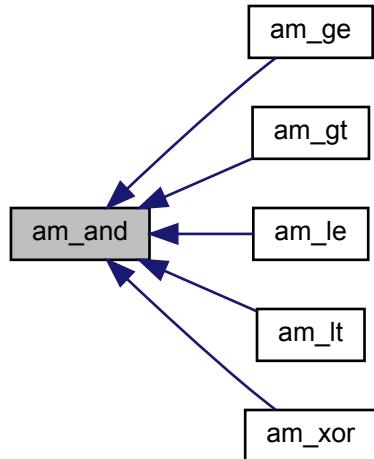
<i>fun</i>	logical value, negative for false, positive for true
------------	--

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

- mlhsInnnerSubst< 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

<i>varargin</i>	elements for chain of equalities
-----------------	----------------------------------

## Return values

<i>fun</i>	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}  $\geq$  varargin{2},varargin{2}  $\geq$  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}  $\geq$  varargin{2},varargin{2}  $\geq$  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 $\geq$ 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

<b>varargin</b>	chain of input parameters
-----------------	---------------------------

##### Return values

<b>fun</b>	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

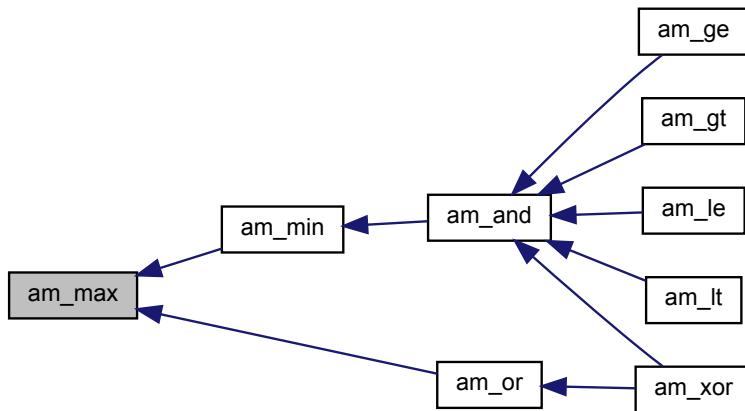
<i>a</i>	first input parameter
<i>b</i>	second input parameter

#### 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

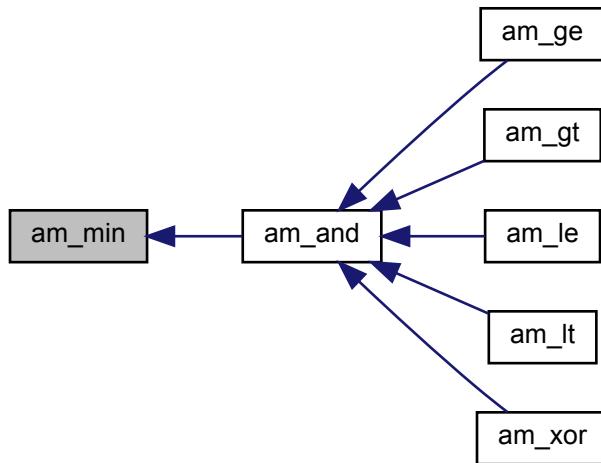
<code>fun</code>	minimum of a and b
------------------	--------------------

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< matlabypesubstitute > [am\\_stepfun](#) (:sym *t*, matlabypesubstitute *tstart*, matlabypesubstitute *vstart*, matlabypesubstitute *tend*, matlabypesubstitute *vend*)

*am\_stepfun is the amici implementation of the step function*

### 11.12.1 Function Documentation

#### 11.12.1.1 am\_stepfun()

```
mlhsInnerSubst< matlabypesubstitute > am_stepfun (
    ::sym t,
    matlabypesubstitute tstart,
    matlabypesubstitute vstart,
    matlabypesubstitute tend,
    matlabypesubstitute 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

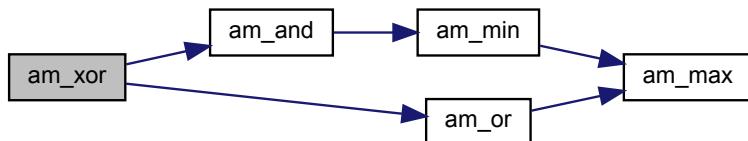
<i>a</i>	first input parameter
<i>b</i>	second input parameter

## 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:

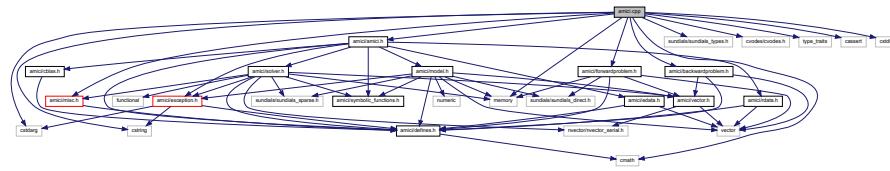


## 11.14 amici.cpp File Reference

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 <cmath>
```

Include dependency graph for amici.cpp:



### Namespaces

- `amici`

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

### Macros

- `#define _USE_MATH_DEFINES`
- `#define M_PI 3.14159265358979323846`

### Functions

- `std::unique_ptr< ReturnData > runAmiciSimulation (Solver &solver, const ExpData *edata, Model &model)`
- `void printErrMsgIdAndTxt (const char *identifier, const char *format,...)`
- `void printWarnErrMsgIdAndTxt (const char *identifier, const char *format,...)`

#### 11.14.1 Macro Definition Documentation

### 11.14.1.1 \_USE\_MATH\_DEFINES

```
#define _USE_MATH_DEFINES
```

MS definition of PI and other constants

Definition at line 23 of file amici.cpp.

### 11.14.1.2 M\_PI

```
#define M_PI 3.14159265358979323846
```

define PI if we still have no definition

Definition at line 27 of file amici.cpp.

## 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

<b>varargin</b>	<pre>amiwrap ( modelname, symfun, tdir, o2flag )</pre> <p><i>Required Parameters for varargin:</i></p> <ul style="list-style-type: none"> <li>• modelname specifies the name of the model which will be later used for the naming of the simulation file</li> <li>• symfun specifies a function which executes model definition see <a href="#">MATLAB Interface</a> for details</li> <li>• tdir target directory where the simulation file should be placed <b>Default:</b> \$AMICIDIR/models/modelname</li> <li>• o2flag boolean whether second order sensitivities should be enabled <b>Default:</b> false</li> </ul>
-----------------	--

## Return values

<i>o2flag</i>	void
---------------	------

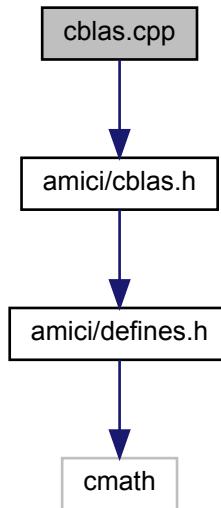
Definition at line 17 of file amiwrap.m.

## 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)

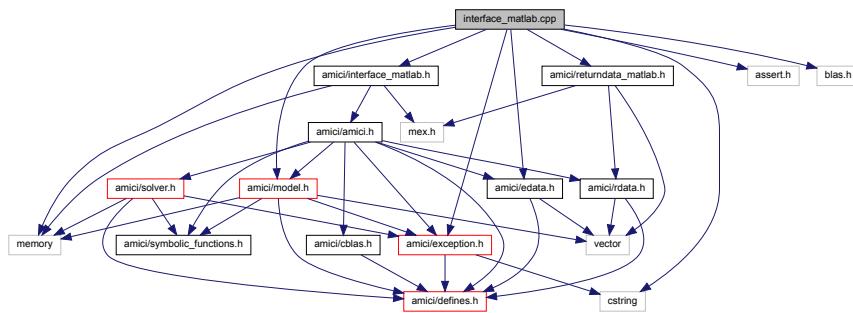
*Compute  $y = a*x + y$ .*

## 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

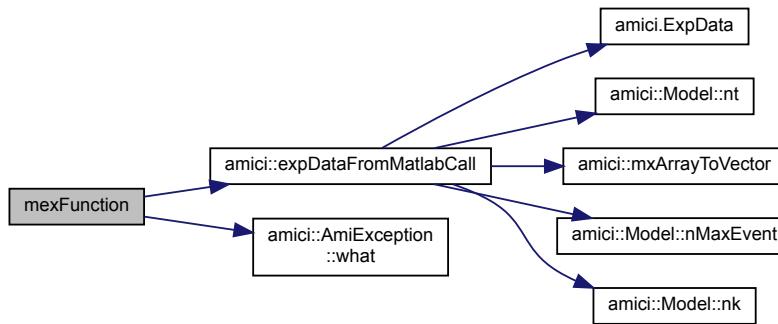
<i>nlhs</i>	number of output arguments of the matlab call
<i>plhs</i>	pointer to the array of output arguments
<i>nrhs</i>	number of input arguments of the matlab call
<i>prhs</i>	pointer to the array of input arguments

**Returns**

```
void
```

Definition at line 517 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**

<i>filename</i>	name of the SBML file (without extension)
<i>modelname</i>	name of the model, this will define the name of the output file (default: input filename)

**Return values**

<i>modelname</i>	void
------------------	------

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[])
    - Integrate the cubic spline function.*

### 11.19.1 Detailed Description

#### Author

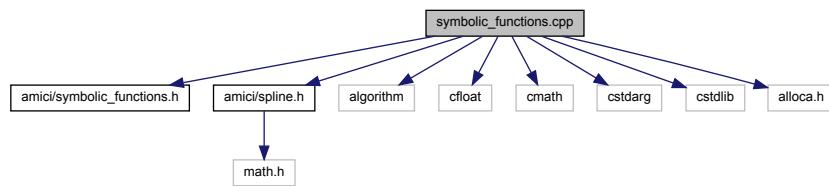
Peter & Nigel, Design Software, 42 Gubberley St, Kenmore, 4069, Australia.

## 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

\_USE\_MATH\_DEFINES  
    amici.cpp, 528

\_\_init\_\_  
    amici::sbml\_import::SbmlImporter, 365

~Model  
    amici::Model, 149

adjInit  
    amici::Solver, 438

adjoint  
    amimodel, 493

allocateSolver  
    amici::Solver, 432

allocateSolverB  
    amici::Solver, 438

am\_and  
    am\_and.m, 515

am\_and.m, 515  
    am\_and, 515

am\_eq  
    am\_eq.m, 517

am\_eq.m, 516  
    am\_eq, 517

am\_ge  
    am\_ge.m, 517

am\_ge.m, 517  
    am\_ge, 517

am\_gt  
    am\_gt.m, 518

am\_gt.m, 518  
    am\_gt, 518

am\_if  
    am\_if.m, 519

am\_if.m, 519  
    am\_if, 519

am\_le  
    am\_le.m, 520

am\_le.m, 520  
    am\_le, 520

am\_lt  
    am\_lt.m, 521

am\_lt.m, 521  
    am\_lt, 521

am\_max  
    am\_max.m, 522

am\_max.m, 522  
    am\_max, 522

am\_min  
    am\_min.m, 523

am\_min.m, 523  
    am\_min, 523

am\_or  
    am\_or.m, 524

am\_or.m, 524  
    am\_or, 524

am\_piecewise  
    am\_piecewise.m, 525

am\_piecewise.m, 525  
    am\_piecewise, 525

am\_stepfun  
    am\_stepfun.m, 526

am\_stepfun.m, 526  
    am\_stepfun, 526

am\_xor  
    am\_xor.m, 527

am\_xor.m, 527  
    am\_xor, 527

AmiException, 76  
    amici::AmiException, 77, 78

AmiVector, 80  
    amici::AmiVector, 80, 81

AmiVectorArray, 88  
    amici::AmiVectorArray, 89

amici, 14  
    amici\_blasCBlasTransToBlasTrans, 66  
    amici\_daxpy, 27  
    amici\_dgemm, 25  
    amici\_dgemv, 24  
    amici\_path, 70  
    BLASLayout, 19  
    BLASTranspose, 19  
    checkFieldNames, 42  
    checkFinite, 32  
    DDspline, 61  
    DDspline\_pos, 61  
    dbl2int, 65  
    deserializeFromChar, 44  
    deserializeFromString, 45  
    dirac, 50  
    Dmax, 53  
    Dmin, 52  
    Dspline, 59  
    Dspline\_pos, 60  
    errMsgIdAndTxt, 70  
    ExpData, 63  
    expDataFromMatlabCall, 31  
    getNaN, 56  
    getReturnDataMatlabFromAmiciCall, 33  
    getValueByIdx, 67  
    heaviside, 51  
    initAndAttachArray, 41  
    initMatlabDiagnosisFields, 35  
    initMatlabReturnFields, 34  
    InternalSensitivityMethod, 21  
    InterpolationType, 21  
    isInf, 55  
    isNaN, 54  
    LinearMultistepMethod, 21  
    LinearSolver, 20  
    log, 50

max, 52  
 min, 51  
 msgIdAndTxtFp, 19  
 mxArrayToVector, 66  
 NewtonStatus, 22  
 NonlinearSolverIteration, 21  
 operator==, 33, 46  
 ParameterScaling, 20  
 pi, 70  
 pos\_pow, 54  
 printErrMsgIdAndTxt, 22  
 printWarnMsgIdAndTxt, 23  
 realtype, 19  
 reorder, 43  
 runAmiciSimulation, 23, 62  
 runAmiciSimulations, 64  
 SecondOrderMode, 20  
 SensitivityMethod, 20  
 SensitivityOrder, 20  
 serializeToChar, 44  
 serializeToStdVec, 45  
 serializeToString, 45  
 setModelData, 28  
 setSolverOptions, 29  
 setValueByld, 68  
 setValueByldRegex, 69  
 setupReturnData, 30  
 seval, 47  
 sign, 57  
 sinteg, 49  
 spline, 46, 57  
 spline\_pos, 58  
 StateOrdering, 21  
 SteadyStateSensitivityMode, 22  
 warnMsgIdAndTxt, 70  
 writeMatlabField0, 36  
 writeMatlabField1, 37  
 writeMatlabField2, 38  
 writeMatlabField3, 39  
 writeMatlabField4, 40  
 amici.cpp, 528  
 \_USE\_MATH\_DEFINES, 528  
 M\_PI, 529  
 amici.plotting, 71  
 amici.sbml\_import, 72  
 amici::AmiException  
 AmiException, 77, 78  
 getBacktrace, 79  
 storeBacktrace, 79  
 what, 78  
 amici::AmiVector  
 AmiVector, 80, 81  
 at, 88  
 data, 82, 83  
 getLength, 86  
 getNVector, 84  
 getVector, 85  
 minus, 86  
 operator=, 81  
 operator[], 87  
 reset, 86  
 set, 87  
 amici::AmiVectorArray  
 AmiVectorArray, 89  
 at, 91  
 data, 90, 91  
 getLength, 93  
 getNVector, 92  
 getNVectorArray, 92  
 operator[], 93  
 reset, 94  
 amici::BackwardProblem  
 BackwardProblem, 95  
 getdJydx, 99  
 getdxBptr, 99  
 gett, 96  
 getwhich, 97  
 getwhichptr, 97  
 getxBptr, 98  
 getxQBptr, 98  
 workBackwardProblem, 96  
 amici::CvodeException  
 CvodeException, 101  
 amici::ExpData  
 checkDataDimension, 124  
 checkEventsDimension, 124  
 checkSigmaPositivity, 125, 126  
 ExpData, 103–106  
 fixedParameters, 127  
 fixedParametersPreequilibration, 127  
 fixedParametersPresimulation, 127  
 getObservedData, 111  
 getObservedDataPtr, 111  
 getObservedDataStdDev, 116  
 getObservedDataStdDevPtr, 116  
 getObservedEvents, 118  
 getObservedEventsPtr, 119  
 getObservedEventsStdDev, 123  
 getObservedEventsStdDevPtr, 123  
 getTimepoint, 109  
 getTimepoints, 108  
 isSetObservedData, 110  
 isSetObservedDataStdDev, 115  
 isSetObservedEvents, 118  
 isSetObservedEventsStdDev, 122  
 nmaxevent, 126  
 nt, 107  
 nytrue, 126  
 nztrue, 126  
 observedData, 127  
 observedDataStdDev, 128  
 observedEvents, 128  
 observedEventsStdDev, 128  
 setObservedData, 109, 110  
 setObservedDataStdDev, 112–114  
 setObservedEvents, 117

setObservedEventsStdDev, 119–121  
setTimepoints, 107  
ts, 127  
amici::ForwardProblem  
edata, 136  
ForwardProblem, 129  
getDJydx, 133  
getDJzdx, 133  
getDiscontinuities, 132  
getNumberOfRoots, 132  
getRHSAtDiscontinuities, 131  
getRHSBeforeDiscontinuities, 132  
getRootCounter, 133  
getRootIndexes, 132  
getStateDerivativePointer, 134  
getStateDerivativeSensitivityPointer, 135  
getStatePointer, 134  
getStateSensitivity, 131  
getStateSensitivityPointer, 134  
getStatesAtDiscontinuities, 131  
getTime, 130  
model, 135  
rdata, 135  
solver, 135  
workForwardProblem, 130  
amici::IDAException  
IDAException, 137  
amici::IntegrationFailure  
error\_code, 138  
IntegrationFailure, 138  
time, 138  
amici::IntegrationFailureB  
error\_code, 140  
IntegrationFailureB, 139  
time, 140  
amici::Model  
~Model, 149  
boost::serialization::serialize, 257  
checkFinite, 214  
clone, 150  
dJrzdsigma, 265  
dJrzdz, 265  
dJydp, 262  
dJydsigma, 264  
dJydy, 264  
dJzdp, 262  
dJzdsigma, 264  
dJzdz, 264  
deltaqb, 263  
deltasx, 262  
deltax, 262  
deltaxB, 263  
drzdp, 266  
drzdx, 265  
dsigmaydp, 261  
dsigmazdp, 262  
dwdp, 267  
dwdx, 266  
dxdotdp, 263  
dydp, 266  
dydx, 266  
dzdp, 265  
dzdx, 265  
fFIM, 213  
fJDiag, 153  
fJSparse, 153  
fJrz, 175, 244  
fJv, 154  
fJy, 173, 243  
fJz, 174, 244  
fchi2, 212  
fdJrzdsigma, 179, 247  
fdJrzdz, 178, 247  
fdJydp, 181  
fdJydsigma, 176, 245  
fdJydx, 182  
fdJydy, 175, 245  
fdJzdp, 184  
fdJzdsigma, 177, 246  
fdJzdx, 184  
fdJzdz, 177, 246  
fdeltaqb, 170, 241  
fdeltasx, 168, 240  
deltax, 167, 238  
deltaxB, 169, 240  
fdrzdp, 166, 237  
fdrzdx, 167, 238  
fsigmaydp, 171, 242  
fsigmazdp, 172, 243  
fdwdp, 208, 248  
fdwdx, 209, 249  
fdx0, 157  
fdxdotdp, 154  
fydp, 160, 233  
fydx, 161, 234  
fdzdp, 165, 236  
fdzdx, 166, 237  
fixedParameters, 268  
fJ, 152  
fres, 211  
froot, 150  
frz, 163, 235  
fsJy, 181  
fsJz, 183  
fsdx0, 159  
fsigmay, 171, 242  
fsigmaz, 172, 243  
fsres, 212  
fsrz, 164, 236  
fstau, 159, 232  
fsx0, 157, 232  
fsx0\_fixedParameters, 158, 231  
fsy, 179  
fsz, 162, 234  
fsz\_tf, 180  
fw, 207, 248

fx0, 155, 230  
 fx0\_fixedParameters, 156, 231  
 fxdot, 151  
 fy, 159, 233  
 fz, 161, 234  
 getFixedParameterById, 196  
 getFixedParameterByName, 196  
 getFixedParameterIds, 227  
 getFixedParameterNames, 219  
 getFixedParameters, 194  
 getInitialStateSensitivities, 203  
 getInitialStates, 203  
 getObservableIds, 228  
 getObservableNames, 219  
 getParameterById, 221  
 getParameterByName, 222  
 getParameterIds, 221  
 getParameterList, 202  
 getParameterNames, 216  
 getParameterScale, 192  
 getParameters, 193  
 getReinitializeFixedParameterInitialStates, 230  
 getSolver, 150  
 getStatelids, 226  
 getStateNames, 217  
 getSteadyStateSensitivityMode, 229  
 getTimepoints, 199  
 getUnscaledParameters, 194  
 getmy, 249  
 getmz, 250  
 getrz, 254  
 getsrz, 256  
 getsx, 253  
 getsz, 255  
 gett, 214  
 getx, 252  
 gety, 251  
 getz, 254  
 h, 267  
 hasFixedParameterIds, 226  
 hasFixedParameterNames, 218  
 hasObservableIds, 228  
 hasObservableNames, 219  
 hasParameterIds, 220  
 hasParameterNames, 215  
 hasStatelids, 225  
 hasStateNames, 217  
 idlist, 261  
 initHeaviside, 186  
 initialize, 185  
 initializeStates, 186  
 isFixedParameterStateReinitializationAllowed, 256  
 J, 263  
 k, 190  
 lbw, 260  
 M, 267  
 Model, 148, 149  
 my, 263  
 mz, 264  
 nMaxEvent, 190  
 ndwdp, 259  
 ndwdx, 259  
 ne, 259  
 nJ, 260  
 nk, 189  
 nmaxevent, 269  
 nnz, 260  
 np, 188  
 plist, 187  
 nt, 191  
 nw, 259  
 nx, 258  
 nxtrue, 258  
 ny, 258  
 nytrue, 258  
 nz, 258  
 nztrue, 259  
 o2mode, 260  
 operator=, 150  
 operator==, 257  
 originalParameters, 268  
 plist, 206  
 plist\_, 268  
 pscale, 269  
 reinitializeFixedParameterInitialStates, 270  
 setFixedParameterById, 197  
 setFixedParameterByName, 198  
 setFixedParameters, 195  
 setFixedParametersByIdRegex, 197  
 setFixedParametersByNameRegex, 199  
 setInitialStateSensitivities, 203  
 setInitialStates, 203  
 setNMaxEvent, 191  
 setParameterById, 223  
 setParameterByName, 224  
 setParameterList, 202  
 setParameterScale, 192, 193  
 setParameters, 194  
 setParametersByIdRegex, 223  
 setParametersByNameRegex, 225  
 setReinitializeFixedParameterInitialStates, 229  
 setSteadyStateSensitivityMode, 229  
 setT0, 205  
 setTimepoints, 200  
 sigmay, 261  
 sigmaz, 261  
 stau, 267  
 steadyStateSensitivityMode, 269  
 sxodata, 268  
 t, 200  
 t0, 204  
 ts, 269  
 tstart, 269  
 ubw, 260  
 unscaleParameters, 206  
 unscaledParameters, 267

updateHeaviside, 213  
updateHeavisideB, 214  
w, 266  
x0data, 268  
z2event, 261  
amici::Model\_DAE  
  fJDiag, 278, 293  
  fJSparse, 276, 277, 291  
  fJSparseB, 277, 292  
  fJB, 275, 291  
  fJv, 279, 280, 293  
  fJvB, 281, 294  
  fdxdotdp, 286, 287, 297  
  fJ, 273, 274, 290  
  fM, 289, 299  
  fqBdot, 286, 296  
  froot, 282, 283, 295  
  fsxdot, 288, 298  
  fxBdot, 285, 296  
  fxdot, 283, 284, 295  
  getSolver, 290  
  Model\_DAE, 272  
amici::Model\_ODE  
  fJDiag, 307, 308, 322  
  fJSparse, 305, 306, 321  
  fJSparseB, 306, 322  
  fJB, 304, 321  
  fJv, 309, 310, 323  
  fJvB, 311, 323  
  fdxdotdp, 317, 318, 326  
  fJ, 302, 303, 320  
  fqBdot, 316, 325  
  froot, 311, 312, 324  
  fsxdot, 319, 326  
  fxBdot, 316, 325  
  fxdot, 314, 315, 324  
  getSolver, 320  
  Model\_ODE, 301, 302  
amici::NewtonFailure  
  error\_code, 329  
  NewtonFailure, 328  
amici::NewtonSolver  
  atol, 334  
  computeNewtonSensis, 332  
  dx, 336  
  getSolver, 330  
  getStep, 331  
  maxlinsteps, 334  
  maxsteps, 334  
  model, 335  
  NewtonSolver, 330  
  prepareLinearSystem, 333  
  rdata, 335  
  rtol, 335  
  solveLinearSystem, 333  
  t, 335  
  x, 336  
  xdot, 335  
    amici::NewtonSolverDense  
      NewtonSolverDense, 337  
      prepareLinearSystem, 338  
      solveLinearSystem, 337  
    amici::NewtonSolverIterative  
      linsolveSPBCG, 341  
      NewtonSolverIterative, 339  
      prepareLinearSystem, 341  
      solveLinearSystem, 340  
    amici::NewtonSolverSparse  
      NewtonSolverSparse, 343  
      prepareLinearSystem, 344  
      solveLinearSystem, 344  
  amici::ReturnData  
    applyChainRuleFactorToSimulationResults, 349  
    boost::serialization::serialize, 349  
    chi2, 357  
    FIM, 353  
    invalidate, 347  
    invalidateLLH, 348  
    J, 350  
    llh, 356  
    ne, 359  
    newton\_maxsteps, 360  
    newton\_numlinsteps, 356  
    newton\_numsteps, 356  
    newton\_status, 355  
    newton\_time, 355  
    nJ, 359  
    nk, 358  
    nmaxevent, 360  
    np, 357  
    nplist, 359  
    nt, 360  
    numerrtestfails, 354  
    numerrtestfailsB, 354  
    numnonlinsolvconvfails, 355  
    numnonlinsolvconvfailsB, 355  
    numrhsevals, 354  
    numrhsevalsB, 354  
    numsteps, 353  
    numstepsB, 354  
    nx, 358  
    nxtrue, 358  
    ny, 358  
    nytrue, 358  
    nz, 359  
    nztrue, 359  
    o2mode, 360  
    order, 355  
    pscale, 360  
    res, 353  
    ReturnData, 347  
    rz, 351  
    s2llh, 357  
    s2rz, 351  
    sensi, 361  
    sensi\_meth, 361

sigmay, 352  
 sigmaz, 350  
 sllh, 357  
 sres, 353  
 srz, 351  
 ssigmay, 353  
 ssigmaz, 351  
 status, 357  
 sx, 352  
 sx0, 356  
 sy, 352  
 sz, 351  
 ts, 350  
 x, 352  
 x0, 356  
 xdot, 350  
 y, 352  
 z, 350  
**amici::SetupFailure**  
 SetupFailure, 395  
**amici::Solver**  
 adjInit, 438  
 allocateSolver, 432  
 allocateSolverB, 438  
 applyQuadTolerancesASA, 454  
 applySensitivityTolerances, 454  
 applyTolerances, 452  
 applyTolerancesASA, 453  
 applyTolerancesFSA, 452  
 band, 441  
 bandB, 441  
 binit, 427  
 boost::serialization::serialize, 455  
 calcICB, 405  
 calcIC, 405  
 clone, 400  
 dense, 440  
 denseB, 440  
 diag, 442  
 diagB, 442  
 getAbsoluteTolerance, 416  
 getAbsoluteToleranceQuadratures, 418  
 getAdjBmem, 451  
 getAdjMallocDone, 451  
 getDiagnosis, 403  
 getDiagnosisB, 403  
 getDky, 438  
 getInternalSensitivityMethod, 426  
 getInterpolationType, 423  
 getLastOrder, 447  
 getLinearMultistepMethod, 421  
 getLinearSolver, 425  
 getMallocDone, 451  
 getMaxSteps, 419  
 getMaxStepsBackwardProblem, 420  
 getModel, 450  
 getNewtonMaxLinearSteps, 412  
 getNewtonMaxSteps, 410  
 getNewtonPreequilibration, 411  
 getNonlinearSolverIteration, 422  
 getNumErrTestFails, 446  
 getNumNonlinSolvConvFails, 447  
 getNumRhsEvals, 446  
 getNumSteps, 445  
 getQuadB, 408  
 getRelativeTolerance, 414  
 getRelativeToleranceQuadratures, 417  
 getRootInfo, 404  
 getSens, 402  
 getSensitivityMethod, 409  
 getSensitivityOrder, 413  
 getStabilityLimitFlag, 424  
 getStateOrdering, 423  
 getB, 408  
 init, 427  
 initializeLinearSolver, 448  
 initializeLinearSolverB, 449  
 interpType, 457  
 ism, 456  
 iter, 457  
 klu, 444  
 kluSetOrdering, 444  
 kluSetOrderingB, 445  
 klub, 445  
 lmm, 456  
 maxsteps, 457  
 nplist, 450  
 nx, 450  
 operator==, 455  
 qbinit, 428  
 quadReInitB, 409  
 quadSStolerancesB, 439  
 relInit, 404  
 relInitB, 408  
 rootInit, 428  
 sensInit1, 429  
 sensRelInit, 405  
 setAbsoluteTolerance, 416  
 setAbsoluteToleranceQuadratures, 418  
 setBandJacFn, 430  
 setBandJacFnB, 431  
 setDenseJacFn, 429  
 setDenseJacFnB, 430  
 setErrHandlerFn, 434  
 setId, 437  
 setInternalSensitivityMethod, 426  
 setInterpolationType, 423  
 setJacTimesVecFn, 430  
 setJacTimesVecFnB, 431  
 setLinearMultistepMethod, 421  
 setLinearSolver, 426  
 setMaxNumSteps, 435  
 setMaxNumStepsB, 436  
 setMaxSteps, 419  
 setMaxStepsBackwardProblem, 420  
 setNewtonMaxLinearSteps, 412

setNewtonMaxSteps, 410  
setNewtonPreequilibration, 411  
setNonlinearSolverIteration, 422  
setQuadErrConB, 434  
setRelativeTolerance, 414  
setRelativeToleranceQuadratures, 417  
setSStolerances, 432  
setSStolerancesB, 439  
setSensErrCon, 433  
setSensParams, 438  
setSensSStolerances, 433  
setSensitivityMethod, 409  
setSensitivityOrder, 413  
setSparseJacFn, 430  
setSparseJacFnB, 431  
setStabLimDet, 436  
setStabLimDetB, 437  
setStabilityLimitFlag, 424  
setStateOrdering, 424  
setStopTime, 407  
setSuppressAlg, 438  
setUserData, 434  
setUserDataB, 435  
setup, 400  
setupAMIB, 401  
solve, 406  
solveB, 407  
solveF, 406  
Solver, 400  
solverMemory, 456  
solverMemoryB, 456  
solverWasCalled, 456  
spbcg, 443  
spbcgB, 443  
spgmr, 442  
spgmrB, 442  
sptfqmr, 443  
sptfqmrB, 444  
turnOffRootFinding, 409  
wrapErrorHandlerFn, 432  
amici::SteadystateProblem  
    applyNewtonsMethod, 459  
    createSteadystateSimSolver, 461  
    getNewtonOutput, 460  
    getSteadystateSimulation, 460  
    SteadystateProblem, 458  
    workSteadyStateProblem, 458  
amici::plotting  
    plotObservableTrajectories, 71  
    plotStateTrajectories, 71  
amici::sbml\_import  
    applyTemplate, 73  
    assignmentRules2observables, 75  
    constantSpeciesToParameters, 75  
    getRuleVars, 75  
    getSymbolicDiagonal, 74  
    getSymbols, 73  
    replaceLogAB, 76  
amici::sbml\_importer  
    \_\_init\_\_, 365  
    checkLibSBMLErrors, 366  
    checkSupport, 371  
    cleanReservedSymbols, 377  
    compileCCode, 384  
    computeModelEquations, 379  
    computeModelEquationsAdjointSensitivites, 382  
    computeModelEquationsForwardSensitivites, 381  
    computeModelEquationsLinearSolver, 380  
    computeModelEquationsObjectiveFunction, 380  
    computeModelEquationsSensitivitesCore, 381  
    generateCCode, 383  
    getFunctionBody, 386  
    getSparseSymLines, 392  
    getSparseSymbols, 378  
    getSymLines, 392  
    getSymbolIDInitializerList, 389  
    getSymbolNameInitializerList, 389  
    loadSBMLFile, 365  
    prepareModelFolder, 383  
    printWithException, 393  
    processCompartments, 373  
    processParameters, 371  
    processReactions, 373  
    processRules, 374  
    processSBML, 370  
    processSpecies, 371  
    processTime, 375  
    processVolumeConversion, 374  
    replaceInAllExpressions, 375  
    replaceSpecialConstants, 377  
    sbml2amici, 366  
    setName, 368  
    setPaths, 369  
    writeCMakeFile, 390  
    writeFunctionFile, 385  
    writeIndexFiles, 385  
    writeModelHeader, 388  
    writeModuleSetup, 391  
    writeSwigFiles, 390  
    writeWrapfunctionsCPP, 387  
    writeWrapfunctionsHeader, 387  
amici\_blasCBlasTransToBlasTrans  
    amici, 66  
amici\_daxpy  
    amici, 27  
amici\_dgemm  
    amici, 25  
amici\_dgemv  
    amici, 24  
amici\_path  
    amici, 70  
amidata, 463  
    amidata, 464  
    condition, 467  
    conditionPreequilibration, 467  
    ne, 465

nk, 465  
 nt, 464  
 ny, 465  
 nz, 465  
 Sigma\_Y, 466  
 Sigma\_Z, 467  
 t, 466  
 Y, 466  
 Z, 466  
 amievent, 468  
   amievent, 468  
   bolus, 469  
   hflag, 470  
   setHflag, 469  
   trigger, 469  
   z, 470  
 amifun, 470  
   amifun, 472  
   argstr, 477  
   cvar, 477  
   deps, 477  
   funstr, 477  
   gccode, 473  
   getArgs, 474  
   getCVar, 475  
   getDeps, 474  
   getNVecs, 475  
   getSensiFlag, 475  
   getSyms, 475  
   nvecs, 478  
   sensiflag, 478  
   strsym, 476  
   strsym\_old, 477  
   sym, 476  
   sym\_noopt, 476  
   writeCcode, 472  
   writeCcode\_sensi, 472  
   writeMcode, 473  
 amimodel, 478  
   adjoint, 493  
   amimodel, 482  
   augmento2, 489  
   augmento2vec, 489  
   cfun, 500  
   checkDeps, 487  
   colptrs, 498  
   colptrsB, 499  
   compileAndLinkModel, 490  
   compileC, 484  
   coptim, 499  
   debug, 493  
   event, 492  
   forward, 493  
   fun, 492  
   funs, 499  
   generateMatlabWrapper, 491  
   generateC, 484  
   generateM, 485  
     getFun, 485  
     HTable, 492  
     id, 496  
     lbw, 497  
     loadOldHashes, 488  
     makeEvents, 486  
     makeSyms, 487  
     maxflag, 502  
     mfuns, 499  
     minflag, 501  
     modelname, 492  
     ndwdp, 502  
     ndwdx, 502  
     nevent, 496  
     ng, 495  
     nk, 495  
     nnz, 497  
     np, 495  
     nw, 502  
     nx, 494  
     nxtrue, 494  
     ny, 494  
     nytrue, 495  
     nz, 496  
     nztrue, 496  
     o2flag, 501  
     param, 500  
     parseModel, 483  
     recompile, 500  
     rowvals, 498  
     rowvalsB, 498  
     sparseidx, 497  
     sparseidxB, 498  
     splineflag, 501  
     sym, 491  
     t0, 493  
     ubw, 497  
     update modelName, 483  
     updateRHS, 482  
     updateWrapPath, 483  
     wrap\_path, 500  
     wtype, 494  
     z2event, 501  
     amioption, 503  
       amioption, 504  
       atol, 505  
       interpType, 507  
       ism, 507  
       iter, 506  
       linsol, 507  
       lmm, 506  
       maxsteps, 505  
       maxstepsB, 506  
       newton\_maxlinsteps, 509  
       newton\_maxsteps, 509  
       newton\_preq, 510  
       nmaxevent, 508  
       ordering, 508

pscale, 510  
quad\_atol, 505  
quad\_rtol, 505  
rtol, 505  
sens\_ind, 506  
sensi, 508  
sensi\_meth, 507  
ss, 508  
stldet, 507  
sx0, 509  
tstart, 506  
x0, 509  
z2event, 510  
amised, 511  
    amised, 511  
    datasym, 513  
    model, 512  
    modelname, 512  
    outputcount, 513  
    sedml, 512  
    varidx, 513  
    varsym, 513  
amiwrap  
    amiwrap.m, 529  
amiwrap.m, 529  
    amiwrap, 529  
applyChainRuleFactorToSimulationResults  
    amici::ReturnData, 349  
applyNewtonsMethod  
    amici::SteadystateProblem, 459  
applyQuadTolerancesASA  
    amici::Solver, 454  
applySensitivityTolerances  
    amici::Solver, 454  
applyTemplate  
    amici::sbml\_import, 73  
applyTolerances  
    amici::Solver, 452  
applyTolerancesASA  
    amici::Solver, 453  
applyTolerancesFSA  
    amici::Solver, 452  
argstr  
    amifun, 477  
assignmentRules2observables  
    amici::sbml\_import, 75  
at  
    amici::AmiVector, 88  
    amici::AmiVectorArray, 91  
atol  
    amici::NewtonSolver, 334  
    amioption, 505  
augmento2  
    amimodel, 489  
augmento2vec  
    amimodel, 489  
  
BLASLayout  
    amici, 19  
BLASTranspose  
    amici, 19  
BackwardProblem, 94  
    amici::BackwardProblem, 95  
band  
    amici::Solver, 441  
bandB  
    amici::Solver, 441  
binit  
    amici::Solver, 427  
bolus  
    amievent, 469  
boost::serialization::serialize  
    amici::Model, 257  
    amici::ReturnData, 349  
    amici::Solver, 455  
calcICB  
    amici::Solver, 405  
calcIC  
    amici::Solver, 405  
cblas.cpp, 530  
cfun  
    amimodel, 500  
checkDataDimension  
    amici::ExpData, 124  
checkDeps  
    amimodel, 487  
checkEventsDimension  
    amici::ExpData, 124  
checkFieldNames  
    amici, 42  
checkFinite  
    amici, 32  
    amici::Model, 214  
checkLibSBMLErrors  
    amici::sbml\_import::SbmlImporter, 366  
checkSigmaPositivity  
    amici::ExpData, 125, 126  
checkSupport  
    amici::sbml\_import::SbmlImporter, 371  
chi2  
    amici::ReturnData, 357  
cleanReservedSymbols  
    amici::sbml\_import::SbmlImporter, 377  
clone  
    amici::Model, 150  
    amici::Solver, 400  
colptrs  
    amimodel, 498  
colptrsB  
    amimodel, 499  
compileAndLinkModel  
    amimodel, 490  
compileCCode  
    amici::sbml\_import::SbmlImporter, 384  
compileC  
    amimodel, 484  
computeModelEquations

amici::sbml\_import::SbmlImporter, 379  
 computeModelEquationsAdjointSensitivites  
     amici::sbml\_import::SbmlImporter, 382  
 computeModelEquationsForwardSensitivites  
     amici::sbml\_import::SbmlImporter, 381  
 computeModelEquationsLinearSolver  
     amici::sbml\_import::SbmlImporter, 380  
 computeModelEquationsObjectiveFunction  
     amici::sbml\_import::SbmlImporter, 380  
 computeModelEquationsSensitivitesCore  
     amici::sbml\_import::SbmlImporter, 381  
 computeNewtonSensis  
     amici::NewtonSolver, 332  
 condition  
     amidata, 467  
 conditionPreequilibration  
     amidata, 467  
 constantSpeciesToParameters  
     amici::sbml\_import, 75  
 coptim  
     amimodel, 499  
 createSteadystateSimSolver  
     amici::SteadystateProblem, 461  
 cvar  
     amifun, 477  
 CvodeException, 100  
     amici::CvodeException, 101  
  
 DDspline  
     amici, 61  
 DDspline\_pos  
     amici, 61  
 dJrdsigma  
     amici::Model, 265  
 dJrzdz  
     amici::Model, 265  
 dJydp  
     amici::Model, 262  
 dJydsigma  
     amici::Model, 264  
 dJydy  
     amici::Model, 264  
 dJzdp  
     amici::Model, 262  
 dJzdsigma  
     amici::Model, 264  
 dJzdz  
     amici::Model, 264  
 data  
     amici::AmiVector, 82, 83  
     amici::AmiVectorArray, 90, 91  
 datasym  
     amised, 513  
 dbl2int  
     amici, 65  
 debug  
     amimodel, 493  
 deltaqB  
     amici::Model, 263  
  
 deltasx  
     amici::Model, 262  
 deltax  
     amici::Model, 262  
 deltaxB  
     amici::Model, 263  
 dense  
     amici::Solver, 440  
 denseB  
     amici::Solver, 440  
 deps  
     amifun, 477  
 deserializeFromChar  
     amici, 44  
 deserializeFromString  
     amici, 45  
 diag  
     amici::Solver, 442  
 diagB  
     amici::Solver, 442  
 dirac  
     amici, 50  
 Dmax  
     amici, 53  
 Dmin  
     amici, 52  
 drzdp  
     amici::Model, 266  
 drzdx  
     amici::Model, 265  
 dsigmaydp  
     amici::Model, 261  
 dsigmazdp  
     amici::Model, 262  
 Dspline  
     amici, 59  
 Dspline\_pos  
     amici, 60  
 dwdp  
     amici::Model, 267  
 dwdx  
     amici::Model, 266  
 dx  
     amici::NewtonSolver, 336  
 dxdotdp  
     amici::Model, 263  
 dydp  
     amici::Model, 266  
 dydx  
     amici::Model, 266  
 dzdp  
     amici::Model, 265  
 dzdx  
     amici::Model, 265  
  
 edata  
     amici::ForwardProblem, 136  
 errMsgIdAndTxt  
     amici, 70

error\_code  
    amici::IntegrationFailure, 138  
    amici::IntegrationFailureB, 140  
    amici::NewtonFailure, 329

event  
    amimodel, 492

ExpData, 101  
    amici, 63  
    amici::ExpData, 103–106

expDataFromMatlabCall  
    amici, 31

fFIM  
    amici::Model, 213

FIM  
    amici::ReturnData, 353

fJDiag  
    amici::Model, 153  
    amici::Model\_DAE, 278, 293  
    amici::Model\_ODE, 307, 308, 322

fJSparse  
    amici::Model, 153  
    amici::Model\_DAE, 276, 277, 291  
    amici::Model\_ODE, 305, 306, 321

fJSparseB  
    amici::Model\_DAE, 277, 292  
    amici::Model\_ODE, 306, 322

fJB  
    amici::Model\_DAE, 275, 291  
    amici::Model\_ODE, 304, 321

fJrz  
    amici::Model, 175, 244

fJv  
    amici::Model, 154  
    amici::Model\_DAE, 279, 280, 293  
    amici::Model\_ODE, 309, 310, 323

fJvB  
    amici::Model\_DAE, 281, 294  
    amici::Model\_ODE, 311, 323

fJy  
    amici::Model, 173, 243

fJz  
    amici::Model, 174, 244

fchi2  
    amici::Model, 212

fdJrzdsigma  
    amici::Model, 179, 247

fdJrzdz  
    amici::Model, 178, 247

fdJydp  
    amici::Model, 181

fdJydsigma  
    amici::Model, 176, 245

fdJydx  
    amici::Model, 182

fdJydy  
    amici::Model, 175, 245

fdJzdp  
    amici::Model, 184

fdJzdsigma  
    amici::Model, 177, 246

fdJzdx  
    amici::Model, 184

fdJzdz  
    amici::Model, 177, 246

fdeltaqB  
    amici::Model, 170, 241

fdeltsx  
    amici::Model, 168, 240

fdeltax  
    amici::Model, 167, 238

fdeltaxB  
    amici::Model, 169, 240

fdrzdp  
    amici::Model, 166, 237

fdrzdx  
    amici::Model, 167, 238

fdsigmaydp  
    amici::Model, 171, 242

fdsigmazdp  
    amici::Model, 172, 243

fdwdp  
    amici::Model, 208, 248

fdwdx  
    amici::Model, 209, 249

fdx0  
    amici::Model, 157

fdxdotdp  
    amici::Model, 154  
    amici::Model\_DAE, 286, 287, 297  
    amici::Model\_ODE, 317, 318, 326

fdydp  
    amici::Model, 160, 233

fdydx  
    amici::Model, 161, 234

fdzdp  
    amici::Model, 165, 236

fdzdx  
    amici::Model, 166, 237

fixedParameters  
    amici::ExpData, 127  
    amici::Model, 268

fixedParametersPreequilibration  
    amici::ExpData, 127

fixedParametersPresimulation  
    amici::ExpData, 127

fJ  
    amici::Model, 152  
    amici::Model\_DAE, 273, 274, 290  
    amici::Model\_ODE, 302, 303, 320

fM  
    amici::Model\_DAE, 289, 299

forward  
    amimodel, 493

ForwardProblem, 128  
    amici::ForwardProblem, 129

fqBdot

amici::Model\_DAE, 286, 296  
 amici::Model\_ODE, 316, 325  
**fres**  
 amici::Model, 211  
**froot**  
 amici::Model, 150  
 amici::Model\_DAE, 282, 283, 295  
 amici::Model\_ODE, 311, 312, 324  
**frz**  
 amici::Model, 163, 235  
**fsJy**  
 amici::Model, 181  
**fsJz**  
 amici::Model, 183  
**fsdx0**  
 amici::Model, 159  
**fsigmay**  
 amici::Model, 171, 242  
**fsigmaz**  
 amici::Model, 172, 243  
**fsres**  
 amici::Model, 212  
**fsrz**  
 amici::Model, 164, 236  
**fstau**  
 amici::Model, 159, 232  
**fsx0**  
 amici::Model, 157, 232  
**fsx0\_fixedParameters**  
 amici::Model, 158, 231  
**fsxdot**  
 amici::Model\_DAE, 288, 298  
 amici::Model\_ODE, 319, 326  
**fsy**  
 amici::Model, 179  
**fsz**  
 amici::Model, 162, 234  
**fsz\_tf**  
 amici::Model, 180  
**fun**  
 amimodel, 492  
**funss**  
 amimodel, 499  
**funstr**  
 amifun, 477  
**fw**  
 amici::Model, 207, 248  
**fx0**  
 amici::Model, 155, 230  
**fx0\_fixedParameters**  
 amici::Model, 156, 231  
**fxBdot**  
 amici::Model\_DAE, 285, 296  
 amici::Model\_ODE, 316, 325  
**fxdot**  
 amici::Model, 151  
 amici::Model\_DAE, 283, 284, 295  
 amici::Model\_ODE, 314, 315, 324  
**fy**  
 amici::Model, 159, 233  
**fz**  
 amici::Model, 161, 234  
**gccode**  
 amifun, 473  
**generateCCode**  
 amici::sbml\_import::SbmlImporter, 383  
**generateMatlabWrapper**  
 amimodel, 491  
**generateC**  
 amimodel, 484  
**generateM**  
 amimodel, 485  
**getAbsoluteTolerance**  
 amici::Solver, 416  
**getAbsoluteToleranceQuadratures**  
 amici::Solver, 418  
**getAdjBmem**  
 amici::Solver, 451  
**getAdjMallocDone**  
 amici::Solver, 451  
**getArgs**  
 amifun, 474  
**getBacktrace**  
 amici::AmiException, 79  
**getCVar**  
 amifun, 475  
**getDjydx**  
 amici::ForwardProblem, 133  
**getDjzdx**  
 amici::ForwardProblem, 133  
**getDepss**  
 amifun, 474  
**getDiagnosis**  
 amici::Solver, 403  
**getDiagnosisB**  
 amici::Solver, 403  
**getDiscontinuities**  
 amici::ForwardProblem, 132  
**getDky**  
 amici::Solver, 438  
**getFixedParameterByld**  
 amici::Model, 196  
**getFixedParameterByName**  
 amici::Model, 196  
**getFixedParameterIds**  
 amici::Model, 227  
**getFixedParameterNames**  
 amici::Model, 219  
**getFixedParameters**  
 amici::Model, 194  
**getFun**  
 amimodel, 485  
**getFunctionBody**  
 amici::sbml\_import::SbmlImporter, 386  
**getInitialStateSensitivities**  
 amici::Model, 203

getInitialStates  
    amici::Model, 203  
getInternalSensitivityMethod  
    amici::Solver, 426  
getInterpolationType  
    amici::Solver, 423  
getLastOrder  
    amici::Solver, 447  
getLength  
    amici::AmiVector, 86  
    amici::AmiVectorArray, 93  
getLinearMultistepMethod  
    amici::Solver, 421  
getLinearSolver  
    amici::Solver, 425  
getMallocDone  
    amici::Solver, 451  
getMaxSteps  
    amici::Solver, 419  
getMaxStepsBackwardProblem  
    amici::Solver, 420  
getModel  
    amici::Solver, 450  
getNVecs  
    amifun, 475  
getNVector  
    amici::AmiVector, 84  
    amici::AmiVectorArray, 92  
getNVectorArray  
    amici::AmiVectorArray, 92  
getNaN  
    amici, 56  
getNewtonMaxLinearSteps  
    amici::Solver, 412  
getNewtonMaxSteps  
    amici::Solver, 410  
getNewtonOutput  
    amici::SteadystateProblem, 460  
getNewtonPreequilibration  
    amici::Solver, 411  
getNonlinearSolverIteration  
    amici::Solver, 422  
getNumErrTestFails  
    amici::Solver, 446  
getNumNonlinSolvConvFails  
    amici::Solver, 447  
getNumRhsEvals  
    amici::Solver, 446  
getNumSteps  
    amici::Solver, 445  
getNumberOfRoots  
    amici::ForwardProblem, 132  
getObservableIds  
    amici::Model, 228  
getObservableNames  
    amici::Model, 219  
getObservedData  
    amici::ExpData, 111  
getObservedDataPtr  
    amici::ExpData, 111  
getObservedDataStdDev  
    amici::ExpData, 116  
getObservedDataStdDevPtr  
    amici::ExpData, 116  
getObservedEvents  
    amici::ExpData, 118  
getObservedEventsPtr  
    amici::ExpData, 119  
getObservedEventsStdDev  
    amici::ExpData, 123  
getObservedEventsStdDevPtr  
    amici::ExpData, 123  
getParameterById  
    amici::Model, 221  
getParameterByName  
    amici::Model, 222  
getParameterIds  
    amici::Model, 221  
getParameterList  
    amici::Model, 202  
getParameterNames  
    amici::Model, 216  
getParameterScale  
    amici::Model, 192  
getParameters  
    amici::Model, 193  
getQuadB  
    amici::Solver, 408  
getRHSAtDiscontinuities  
    amici::ForwardProblem, 131  
getRHSBeforeDiscontinuities  
    amici::ForwardProblem, 132  
getReinitializeFixedParameterInitialStates  
    amici::Model, 230  
getRelativeTolerance  
    amici::Solver, 414  
getRelativeToleranceQuadratures  
    amici::Solver, 417  
getReturnDataMatlabFromAmiciCall  
    amici, 33  
getRootCounter  
    amici::ForwardProblem, 133  
getRootIndexes  
    amici::ForwardProblem, 132  
getRootInfo  
    amici::Solver, 404  
getRuleVars  
    amici::sbml\_import, 75  
getSens  
    amici::Solver, 402  
getSensiFlag  
    amifun, 475  
getSensitivityMethod  
    amici::Solver, 409  
getSensitivityOrder  
    amici::Solver, 413

getSolver  
 amici::Model, 150  
 amici::Model\_DAE, 290  
 amici::Model\_ODE, 320  
 amici::NewtonSolver, 330

getSparseSymLines  
 amici::sbml\_import::SbmlImporter, 392

getSparseSymbols  
 amici::sbml\_import::SbmlImporter, 378

getStabilityLimitFlag  
 amici::Solver, 424

getStateDerivativePointer  
 amici::ForwardProblem, 134

getStateDerivativeSensitivityPointer  
 amici::ForwardProblem, 135

getStateIds  
 amici::Model, 226

getStateNames  
 amici::Model, 217

getStateOrdering  
 amici::Solver, 423

getStatePointer  
 amici::ForwardProblem, 134

getStateSensitivity  
 amici::ForwardProblem, 131

getStateSensitivityPointer  
 amici::ForwardProblem, 134

getStatesAtDiscontinuities  
 amici::ForwardProblem, 131

getSteadyStateSensitivityMode  
 amici::Model, 229

getSteadystateSimulation  
 amici::SteadystateProblem, 460

getStep  
 amici::NewtonSolver, 331

getSymLines  
 amici::sbml\_import::SbmlImporter, 392

getSymbolIDInitializerList  
 amici::sbml\_import::SbmlImporter, 389

getSymbolNameInitializerList  
 amici::sbml\_import::SbmlImporter, 389

getSymbolicDiagonal  
 amici::sbml\_import, 74

getSymbols  
 amici::sbml\_import, 73

getSyms  
 amifun, 475

getTime  
 amici::ForwardProblem, 130

getTimepoint  
 amici::ExpData, 109

getTimepoints  
 amici::ExpData, 108  
 amici::Model, 199

getUnscaledParameters  
 amici::Model, 194

getValueById  
 amici, 67

getVector  
 amici::AmiVector, 85

getB  
 amici::Solver, 408

getdJydx  
 amici::BackwardProblem, 99

getdxBptr  
 amici::BackwardProblem, 99

getmy  
 amici::Model, 249

getmz  
 amici::Model, 250

getoptimized  
 optsym, 515

getrz  
 amici::Model, 254

getsrz  
 amici::Model, 256

getsx  
 amici::Model, 253

getsz  
 amici::Model, 255

gett  
 amici::BackwardProblem, 96  
 amici::Model, 214

getwhich  
 amici::BackwardProblem, 97

getwhichptr  
 amici::BackwardProblem, 97

getx  
 amici::Model, 252

getxBptr  
 amici::BackwardProblem, 98

getxQBptr  
 amici::BackwardProblem, 98

gety  
 amici::Model, 251

getz  
 amici::Model, 254

h  
 amici::Model, 267

HTable  
 amimodel, 492

hasFixedParameterIds  
 amici::Model, 226

hasFixedParameterNames  
 amici::Model, 218

hasObservableIds  
 amici::Model, 228

hasObservableNames  
 amici::Model, 219

hasParameterIds  
 amici::Model, 220

hasParameterNames  
 amici::Model, 215

hasStatelids  
 amici::Model, 225

hasStateNames

amici::Model, 217  
heaviside  
    amici, 51  
hflag  
    amievent, 470  
  
IDAException, 136  
    amici::IDAException, 137  
id  
    amimodel, 496  
idlist  
    amici::Model, 261  
init  
    amici::Solver, 427  
initAndAttachArray  
    amici, 41  
initHeaviside  
    amici::Model, 186  
initMatlabDiagnosisFields  
    amici, 35  
initMatlabReturnFields  
    amici, 34  
initialize  
    amici::Model, 185  
initializeLinearSolver  
    amici::Solver, 448  
initializeLinearSolverB  
    amici::Solver, 449  
initializeStates  
    amici::Model, 186  
IntegrationFailure, 137  
    amici::IntegrationFailure, 138  
IntegrationFailureB, 139  
    amici::IntegrationFailureB, 139  
interface\_matlab.cpp, 531  
    mexFunction, 532  
InternalSensitivityMethod  
    amici, 21  
interpType  
    amici::Solver, 457  
    amioption, 507  
InterpolationType  
    amici, 21  
invalidate  
    amici::ReturnData, 347  
invalidateLLH  
    amici::ReturnData, 348  
isFixedParameterStateReinitializationAllowed  
    amici::Model, 256  
isInf  
    amici, 55  
isNaN  
    amici, 54  
isSetObservedData  
    amici::ExpData, 110  
isSetObservedDataStdDev  
    amici::ExpData, 115  
isSetObservedEvents  
    amici::ExpData, 118  
  
isSetObservedEventsStdDev  
    amici::ExpData, 122  
ism  
    amici::Solver, 456  
    amioption, 507  
iter  
    amici::Solver, 457  
    amioption, 506  
  
J  
    amici::Model, 263  
    amici::ReturnData, 350  
  
k  
    amici::Model, 190  
klu  
    amici::Solver, 444  
kluSetOrdering  
    amici::Solver, 444  
kluSetOrderingB  
    amici::Solver, 445  
kluB  
    amici::Solver, 445  
  
lbw  
    amici::Model, 260  
    amimodel, 497  
LinearMultistepMethod  
    amici, 21  
LinearSolver  
    amici, 20  
linsol  
    amioption, 507  
linsolveSPBCG  
    amici::NewtonSolverIterative, 341  
llh  
    amici::ReturnData, 356  
Imm  
    amici::Solver, 456  
    amioption, 506  
loadOldHashes  
    amimodel, 488  
loadSBMLFile  
    amici::sbml\_import::SbmlImporter, 365  
log  
    amici, 50  
  
M  
    amici::Model, 267  
M\_PI  
    amici.cpp, 529  
makeEvents  
    amimodel, 486  
makeSyms  
    amimodel, 487  
max  
    amici, 52  
maxflag  
    amimodel, 502

maxlinsteps  
     amici::NewtonSolver, 334  
 maxsteps  
     amici::NewtonSolver, 334  
     amici::Solver, 457  
     amioption, 505  
 maxstepsB  
     amioption, 506  
 mexFunction  
     interface\_matlab.cpp, 532  
 mfun  
     amimodel, 499  
 min  
     amici, 51  
 minflag  
     amimodel, 501  
 minus  
     amici::AmiVector, 86  
 Model, 140  
     amici::Model, 148, 149  
 model  
     amici::ForwardProblem, 135  
     amici::NewtonSolver, 335  
     amised, 512  
 Model\_DAE, 270  
     amici::Model\_DAE, 272  
 Model\_ODE, 299  
     amici::Model\_ODE, 301, 302  
 modelname  
     amimodel, 492  
     amised, 512  
 msgIdAndTxtFp  
     amici, 19  
 mxArrayToVector  
     amici, 66  
 my  
     amici::Model, 263  
 mz  
     amici::Model, 264  
 nMaxEvent  
     amici::Model, 190  
 ndwdp  
     amici::Model, 259  
     amimodel, 502  
 ndwdx  
     amici::Model, 259  
     amimodel, 502  
 ne  
     amici::Model, 259  
     amici::ReturnData, 359  
     amidata, 465  
 nevent  
     amimodel, 496  
 newton\_maxlinsteps  
     amioption, 509  
 newton\_maxsteps  
     amici::ReturnData, 360  
     amioption, 509  
     newton\_numlinsteps  
         amici::ReturnData, 356  
 newton\_numsteps  
     amici::ReturnData, 356  
 newton\_preeq  
     amioption, 510  
 newton\_status  
     amici::ReturnData, 355  
 newton\_time  
     amici::ReturnData, 355  
 NewtonFailure, 327  
     amici::NewtonFailure, 328  
 NewtonSolver, 329  
     amici::NewtonSolver, 330  
 NewtonSolverDense, 336  
     amici::NewtonSolverDense, 337  
 NewtonSolverIterative, 339  
     amici::NewtonSolverIterative, 339  
 NewtonSolverSparse, 343  
     amici::NewtonSolverSparse, 343  
 NewtonStatus  
     amici, 22  
 ng  
     amimodel, 495  
 nJ  
     amici::Model, 260  
     amici::ReturnData, 359  
 nk  
     amici::Model, 189  
     amici::ReturnData, 358  
     amidata, 465  
     amimodel, 495  
 nmaxevent  
     amici::ExpData, 126  
     amici::Model, 269  
     amici::ReturnData, 360  
     amioption, 508  
 nnz  
     amici::Model, 260  
     amimodel, 497  
 NonlinearSolverIteration  
     amici, 21  
 np  
     amici::Model, 188  
     amici::ReturnData, 357  
     amimodel, 495  
 nplist  
     amici::Model, 187  
     amici::ReturnData, 359  
     amici::Solver, 450  
 nt  
     amici::ExpData, 107  
     amici::Model, 191  
     amici::ReturnData, 360  
     amidata, 464  
 numerertestfails  
     amici::ReturnData, 354  
 numerertestfailsB

amici::ReturnData, 354  
numnonlinlinsolvconvfails  
    amici::ReturnData, 355  
numnonlinlinsolvconvfailsB  
    amici::ReturnData, 355  
numrhsevals  
    amici::ReturnData, 354  
numrhsevalsB  
    amici::ReturnData, 354  
numsteps  
    amici::ReturnData, 353  
numstepsB  
    amici::ReturnData, 354  
nvecs  
    amifun, 478  
nw  
    amici::Model, 259  
    amimodel, 502  
nx  
    amici::Model, 258  
    amici::ReturnData, 358  
    amici::Solver, 450  
    amimodel, 494  
nxtrue  
    amici::Model, 258  
    amici::ReturnData, 358  
    amimodel, 494  
ny  
    amici::Model, 258  
    amici::ReturnData, 358  
    amidata, 465  
    amimodel, 494  
nytrue  
    amici::ExpData, 126  
    amici::Model, 258  
    amici::ReturnData, 358  
    amimodel, 495  
nz  
    amici::Model, 258  
    amici::ReturnData, 359  
    amidata, 465  
    amimodel, 496  
nztrue  
    amici::ExpData, 126  
    amici::Model, 259  
    amici::ReturnData, 359  
    amimodel, 496  
o2flag  
    amimodel, 501  
o2mode  
    amici::Model, 260  
    amici::ReturnData, 360  
observedData  
    amici::ExpData, 127  
observedDataStdDev  
    amici::ExpData, 128  
observedEvents  
    amici::ExpData, 128  
    observedEventsStdDev  
        amici::ExpData, 128  
operator=  
    amici::AmiVector, 81  
    amici::Model, 150  
operator==  
    amici, 33, 46  
    amici::Model, 257  
    amici::Solver, 455  
operator[]  
    amici::AmiVector, 87  
    amici::AmiVectorArray, 93  
optsym, 514  
    getoptimized, 515  
    optsym, 514  
order  
    amici::ReturnData, 355  
ordering  
    amioption, 508  
originalParameters  
    amici::Model, 268  
outputcount  
    amised, 513  
param  
    amimodel, 500  
ParameterScaling  
    amici, 20  
parseModel  
    amimodel, 483  
pi  
    amici, 70  
plist  
    amici::Model, 206  
plist\_  
    amici::Model, 268  
plotObservableTrajectories  
    amici::plotting, 71  
plotStateTrajectories  
    amici::plotting, 71  
pos\_pow  
    amici, 54  
prepareLinearSystem  
    amici::NewtonSolver, 333  
    amici::NewtonSolverDense, 338  
    amici::NewtonSolverIterative, 341  
    amici::NewtonSolverSparse, 344  
prepareModelFolder  
    amici::sbml\_import::SbmlImporter, 383  
printErrMsgIdAndTxt  
    amici, 22  
printWarnMsgIdAndTxt  
    amici, 23  
printWithException  
    amici::sbml\_import::SbmlImporter, 393  
processCompartments  
    amici::sbml\_import::SbmlImporter, 373  
processParameters  
    amici::sbml\_import::SbmlImporter, 371

**processReactions**  
 amici::sbml\_import::SbmlImporter, 373

**processRules**  
 amici::sbml\_import::SbmlImporter, 374

**processSBML**  
 amici::sbml\_import::SbmlImporter, 370

**processSpecies**  
 amici::sbml\_import::SbmlImporter, 371

**processTime**  
 amici::sbml\_import::SbmlImporter, 375

**processVolumeConversion**  
 amici::sbml\_import::SbmlImporter, 374

**pscale**  
 amici::Model, 269  
 amici::ReturnData, 360  
 amioption, 510

**qbinit**  
 amici::Solver, 428

**quad\_atol**  
 amioption, 505

**quad\_rtol**  
 amioption, 505

**quadReInitB**  
 amici::Solver, 409

**quadSStolerancesB**  
 amici::Solver, 439

**rdata**  
 amici::ForwardProblem, 135  
 amici::NewtonSolver, 335

**reInit**  
 amici::Solver, 404

**reInitB**  
 amici::Solver, 408

**realtype**  
 amici, 19

**recompile**  
 amimodel, 500

**reinitializeFixedParameterInitialStates**  
 amici::Model, 270

**reorder**  
 amici, 43

**replaceInAllExpressions**  
 amici::sbml\_import::SbmlImporter, 375

**replaceLogAB**  
 amici::sbml\_import, 76

**replaceSpecialConstants**  
 amici::sbml\_import::SbmlImporter, 377

**res**  
 amici::ReturnData, 353

**reset**  
 amici::AmiVector, 86  
 amici::AmiVectorArray, 94

**ReturnData**, 345  
 amici::ReturnData, 347

**rootInit**  
 amici::Solver, 428

**rowvals**  
 amimodel, 498  
**rowvalsB**  
 amimodel, 498

**rtol**  
 amici::NewtonSolver, 335  
 amioption, 505

**runAmiciSimulation**  
 amici, 23, 62

**runAmiciSimulations**  
 amici, 64

**rz**  
 amici::ReturnData, 351

**s2lh**  
 amici::ReturnData, 357

**s2rz**  
 amici::ReturnData, 351

**SBML2AMICI.m**, 533  
 SBML2AMICI, 533

**SBML2AMICI**  
 SBML2AMICI.m, 533

**SBMLError**, 361

**sbml2amici**  
 amici::sbml\_import::SbmlImporter, 366

**SbmlImporter**, 362

**SecondOrderMode**  
 amici, 20

**sedml**  
 amised, 512

**sens\_ind**  
 amioption, 506

**sensInit1**  
 amici::Solver, 429

**sensReInit**  
 amici::Solver, 405

**sensi**  
 amici::ReturnData, 361  
 amioption, 508

**sensi\_meth**  
 amici::ReturnData, 361  
 amioption, 507

**sensiflag**  
 amifun, 478

**SensitivityMethod**  
 amici, 20

**SensitivityOrder**  
 amici, 20

**serializeToChar**  
 amici, 44

**serializeToStdVec**  
 amici, 45

**serializeToString**  
 amici, 45

**set**  
 amici::AmiVector, 87

**setAbsoluteTolerance**  
 amici::Solver, 416

**setAbsoluteToleranceQuadratures**  
 amici::Solver, 418

setBandJacFn  
    amici::Solver, 430  
setBandJacFnB  
    amici::Solver, 431  
setDenseJacFn  
    amici::Solver, 429  
setDenseJacFnB  
    amici::Solver, 430  
setErrHandlerFn  
    amici::Solver, 434  
setFixedParameterById  
    amici::Model, 197  
setFixedParameterByName  
    amici::Model, 198  
setFixedParameters  
    amici::Model, 195  
setFixedParametersByIdRegex  
    amici::Model, 197  
setFixedParametersByNameRegex  
    amici::Model, 199  
setHflag  
    amevent, 469  
setId  
    amici::Solver, 437  
setInitialStateSensitivities  
    amici::Model, 203  
setInitialStates  
    amici::Model, 203  
setInternalSensitivityMethod  
    amici::Solver, 426  
setInterpolationType  
    amici::Solver, 423  
setJacTimesVecFn  
    amici::Solver, 430  
setJacTimesVecFnB  
    amici::Solver, 431  
setLinearMultistepMethod  
    amici::Solver, 421  
setLinearSolver  
    amici::Solver, 426  
setMaxNumSteps  
    amici::Solver, 435  
setMaxNumStepsB  
    amici::Solver, 436  
setMaxSteps  
    amici::Solver, 419  
setMaxStepsBackwardProblem  
    amici::Solver, 420  
setModelData  
    amici, 28  
setNMaxEvent  
    amici::Model, 191  
setName  
    amici::sbml\_import::SbmlImporter, 368  
setNewtonMaxLinearSteps  
    amici::Solver, 412  
setNewtonMaxSteps  
    amici::Solver, 410  
setNewtonPreequilibration  
    amici::Solver, 411  
setNonlinearSolverIteration  
    amici::Solver, 422  
setObservedData  
    amici::ExpData, 109, 110  
setObservedDataStdDev  
    amici::ExpData, 112–114  
setObservedEvents  
    amici::ExpData, 117  
setObservedEventsStdDev  
    amici::ExpData, 119–121  
setParameterById  
    amici::Model, 223  
setParameterByName  
    amici::Model, 224  
setParameterList  
    amici::Model, 202  
setParameterScale  
    amici::Model, 192, 193  
setParameters  
    amici::Model, 194  
setParametersByIdRegex  
    amici::Model, 223  
setParametersByNameRegex  
    amici::Model, 225  
setPaths  
    amici::sbml\_import::SbmlImporter, 369  
setQuadErrConB  
    amici::Solver, 434  
setReinitializeFixedParameterInitialStates  
    amici::Model, 229  
setRelativeTolerance  
    amici::Solver, 414  
setRelativeToleranceQuadratures  
    amici::Solver, 417  
setSStolerances  
    amici::Solver, 432  
setSStolerancesB  
    amici::Solver, 439  
setSensErrCon  
    amici::Solver, 433  
setSensParams  
    amici::Solver, 438  
setSensSStolerances  
    amici::Solver, 433  
setSensitivityMethod  
    amici::Solver, 409  
setSensitivityOrder  
    amici::Solver, 413  
setSolverOptions  
    amici, 29  
setSparseJacFn  
    amici::Solver, 430  
setSparseJacFnB  
    amici::Solver, 431  
setStabLimDet  
    amici::Solver, 436

setStabLimDetB  
     amici::Solver, 437  
 setStabilityLimitFlag  
     amici::Solver, 424  
 setStateOrdering  
     amici::Solver, 424  
 setSteadyStateSensitivityMode  
     amici::Model, 229  
 setStopTime  
     amici::Solver, 407  
 setSuppressAlg  
     amici::Solver, 438  
 setT0  
     amici::Model, 205  
 setTimepoints  
     amici::ExpData, 107  
     amici::Model, 200  
 setUserData  
     amici::Solver, 434  
 setUserDataB  
     amici::Solver, 435  
 setValueById  
     amici, 68  
 setValueByIdRegex  
     amici, 69  
 setup  
     amici::Solver, 400  
 setupAMIB  
     amici::Solver, 401  
 SetupFailure, 395  
     amici::SetupFailure, 395  
 setupReturnData  
     amici, 30  
 seval  
     amici, 47  
 Sigma\_Y  
     amidata, 466  
 Sigma\_Z  
     amidata, 467  
 sigmay  
     amici::Model, 261  
     amici::ReturnData, 352  
 sigmaz  
     amici::Model, 261  
     amici::ReturnData, 350  
 sign  
     amici, 57  
 sinteg  
     amici, 49  
 sllh  
     amici::ReturnData, 357  
 solve  
     amici::Solver, 406  
 solveLinearSystem  
     amici::NewtonSolver, 333  
     amici::NewtonSolverDense, 337  
     amici::NewtonSolverIterative, 340  
     amici::NewtonSolverSparse, 344  
 solveB  
     amici::Solver, 407  
 solveF  
     amici::Solver, 406  
 Solver, 396  
     amici::Solver, 400  
 solver  
     amici::ForwardProblem, 135  
 solverMemory  
     amici::Solver, 456  
 solverMemoryB  
     amici::Solver, 456  
 solverWasCalled  
     amici::Solver, 456  
 sparseidx  
     amimodel, 497  
 sparseidxB  
     amimodel, 498  
 spbcg  
     amici::Solver, 443  
 spbcgB  
     amici::Solver, 443  
 spgmr  
     amici::Solver, 442  
 spgmrB  
     amici::Solver, 442  
 spline  
     amici, 46, 57  
 spline.cpp, 534  
 spline\_pos  
     amici, 58  
 splineflag  
     amimodel, 501  
 sptfqmr  
     amici::Solver, 443  
 sptfqmrB  
     amici::Solver, 444  
 sres  
     amici::ReturnData, 353  
 srz  
     amici::ReturnData, 351  
 ss  
     amioption, 508  
 ssigmay  
     amici::ReturnData, 353  
 ssigmaz  
     amici::ReturnData, 351  
 StateOrdering  
     amici, 21  
 status  
     amici::ReturnData, 357  
 stau  
     amici::Model, 267  
 SteadyStateSensitivityMode  
     amici, 22  
 steadyStateSensitivityMode  
     amici::Model, 269  
 SteadystateProblem, 457

amici::SteadystateProblem, 458  
stldet  
    amioption, 507  
storeBacktrace  
    amici::AmiException, 79  
strsym  
    amifun, 476  
strsym\_old  
    amifun, 477  
sx  
    amici::ReturnData, 352  
sx0  
    amici::ReturnData, 356  
    amioption, 509  
sx0data  
    amici::Model, 268  
sy  
    amici::ReturnData, 352  
sym  
    amifun, 476  
    amimodel, 491  
sym\_noopt  
    amifun, 476  
symbolic\_functions.cpp, 534  
sz  
    amici::ReturnData, 351  
  
t  
    amici::Model, 200  
    amici::NewtonSolver, 335  
    amidata, 466  
t0  
    amici::Model, 204  
    amimodel, 493  
TemplateAmici, 394  
time  
    amici::IntegrationFailure, 138  
    amici::IntegrationFailureB, 140  
trigger  
    amevent, 469  
ts  
    amici::ExpData, 127  
    amici::Model, 269  
    amici::ReturnData, 350  
tstart  
    amici::Model, 269  
    amioption, 506  
turnOffRootFinding  
    amici::Solver, 409  
  
ubw  
    amici::Model, 260  
    amimodel, 497  
unscaleParameters  
    amici::Model, 206  
unscaledParameters  
    amici::Model, 267  
updateHeaviside  
    amici::Model, 213  
updateHeavisideB  
    amici::Model, 214  
updatemodelName  
    amimodel, 483  
updateRHS  
    amimodel, 482  
updateWrapPath  
    amimodel, 483  
  
varidx  
    amised, 513  
varsym  
    amised, 513  
  
w  
    amici::Model, 266  
warnMsgIdAndTxt  
    amici, 70  
what  
    amici::AmiException, 78  
workBackwardProblem  
    amici::BackwardProblem, 96  
workForwardProblem  
    amici::ForwardProblem, 130  
workSteadyStateProblem  
    amici::SteadystateProblem, 458  
wrap\_path  
    amimodel, 500  
wrapErrorHandlerFn  
    amici::Solver, 432  
writeCMakeFile  
    amici::sbml\_import::SbmlImporter, 390  
writeCcode  
    amifun, 472  
writeCcode\_sensi  
    amifun, 472  
writeFunctionFile  
    amici::sbml\_import::SbmlImporter, 385  
writeIndexFiles  
    amici::sbml\_import::SbmlImporter, 385  
writeMatlabField0  
    amici, 36  
writeMatlabField1  
    amici, 37  
writeMatlabField2  
    amici, 38  
writeMatlabField3  
    amici, 39  
writeMatlabField4  
    amici, 40  
writeMcode  
    amifun, 473  
writeModelHeader  
    amici::sbml\_import::SbmlImporter, 388  
writeModuleSetup  
    amici::sbml\_import::SbmlImporter, 391  
writeSwigFiles  
    amici::sbml\_import::SbmlImporter, 390  
writeWrapfunctionsCPP

amici::sbml\_import::SbmlImporter, [387](#)  
writeWrapfunctionsHeader  
    amici::sbml\_import::SbmlImporter, [387](#)  
wtype  
    amimodel, [494](#)

x  
    amici::NewtonSolver, [336](#)  
    amici::ReturnData, [352](#)

x0  
    amici::ReturnData, [356](#)  
    amioption, [509](#)

x0data  
    amici::Model, [268](#)

xdot  
    amici::NewtonSolver, [335](#)  
    amici::ReturnData, [350](#)

Y  
    amidata, [466](#)

y  
    amici::ReturnData, [352](#)

Z  
    amidata, [466](#)

z  
    amici::ReturnData, [350](#)  
    amievent, [470](#)

z2event  
    amici::Model, [261](#)  
    amimodel, [501](#)  
    amioption, [510](#)