

AMICI

Generated by Doxygen 1.8.13

## Contents

<b>1</b>	<b>AMICI 0.1 General Documentation</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Availability . . . . .	2
1.3	Installation . . . . .	3
<b>2</b>	<b>How to contribute</b>	<b>4</b>
<b>3</b>	<b>SBMLImporter</b>	<b>4</b>
<b>4</b>	<b>Model Definition &amp; Simulation</b>	<b>4</b>
4.1	Model Definition . . . . .	4
4.2	Model Compilation . . . . .	8
4.3	Model Simulation . . . . .	8
<b>5</b>	<b>Code Organization</b>	<b>11</b>
<b>6</b>	<b>Using AMICI-generated code outside Matlab</b>	<b>12</b>
<b>7</b>	<b>Hierarchical Index</b>	<b>13</b>
7.1	Class Hierarchy . . . . .	13
<b>8</b>	<b>Class Index</b>	<b>14</b>
8.1	Class List . . . . .	14
<b>9</b>	<b>Class Documentation</b>	<b>15</b>
9.1	amidata Class Reference . . . . .	15
9.2	amievent Class Reference . . . . .	20
9.3	amifun Class Reference . . . . .	23
9.4	amimodel Class Reference . . . . .	34
9.5	amioption Class Reference . . . . .	57
9.6	amised Class Reference . . . . .	66
9.7	BackwardProblem Class Reference . . . . .	70
9.8	ExpData Class Reference . . . . .	75

9.9	ForwardProblem Class Reference	77
9.10	Model Class Reference	95
9.11	modelTest Class Reference	145
9.12	NewtonSolver Class Reference	146
9.13	NewtonSolverDense Class Reference	148
9.14	NewtonSolverIterative Class Reference	150
9.15	NewtonSolverSparse Class Reference	152
9.16	optsym Class Reference	154
9.17	ReturnData Class Reference	155
9.18	ReturnDataMatlab Class Reference	177
9.19	SBMLode Class Reference	182
9.20	Solver Class Reference	189
9.21	SteadystateProblem Class Reference	194
9.22	TempData Class Reference	200
9.23	UserData Class Reference	218
<b>10</b>	<b>File Documentation</b>	<b>226</b>
10.1	amiwrap.m File Reference	226
10.2	SBML2AMICI.m File Reference	227
10.3	SBMLImporter/computeBracketLevel.m File Reference	228
10.4	src/amici.cpp File Reference	229
10.5	src/amici_interface_cpp.cpp File Reference	232
10.6	src/amici_interface_matlab.cpp File Reference	236
10.7	src/spline.cpp File Reference	242
10.8	src/symbolic_functions.cpp File Reference	247
10.9	symbolic/am_and.m File Reference	259
10.10	symbolic/am_eq.m File Reference	260
10.11	symbolic/am_ge.m File Reference	261
10.12	symbolic/am_gt.m File Reference	262
10.13	symbolic/am_if.m File Reference	263
10.14	symbolic/am_le.m File Reference	264
10.15	symbolic/am_lt.m File Reference	265
10.16	symbolic/am_max.m File Reference	266
10.17	symbolic/am_min.m File Reference	266
10.18	symbolic/am_or.m File Reference	267
10.19	symbolic/am_piecewise.m File Reference	268
10.20	symbolic/am_stepfun.m File Reference	269
10.21	symbolic/am_xor.m File Reference	270
10.22	tests/cpputest/createTestingData.m File Reference	271
10.23	tests/cpputest/wrapTestModels.m File Reference	271

## 1 AMICI 0.1 General Documentation

### 1.1 Introduction

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

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

### 1.2 Availability

The sources for AMICI are accessible as

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

Once you've obtained your copy check out the [Installation](#)

#### 1.2.1 Obtaining AMICI via the GIT versioning system

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

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

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

### 1.2.2 License Conditions

This software is available under the [BSD license](#)

Copyright (c) 2015, Fabian Fröhlich and Jan Hasenauer All rights reserved.

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

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 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.

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.

## 1.3 Installation

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

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

```
installToolbox.m
```

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

```
savepath
```

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

```
mex -setup c
```

For a list of supported compilers we refer to the mathworks documentation: [mathworks.de](#)

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

AMICI uses the following packages from SUNDIALS:

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

### IDAS

AMICI uses the following packages from SuiteSparse:

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

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

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

## 2 How to contribute

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

### Making code changes

When making code changes:

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

## 3 SBMLImporter

MATLAB toolbox to generate ODE models from SBML files

## 4 Model Definition & Simulation

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

### 4.1 Model Definition

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

#### 4.1.1 Header

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

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

#### 4.1.2 Options

Set the options by specifying the respective field of the modelstruct

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

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

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

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

#### 4.1.3 States

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

```
syms state1 state2 state3
```

Create the state vector containing all states:

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

#### 4.1.4 Parameters

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

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

Create the parameters vector

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

#### 4.1.5 Constants

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

```
syms const1 const2
```

Create the parameters vector

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

#### 4.1.6 Differential Equation

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

```
syms t
```

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

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

or

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

The specification of f or xdot may depend on [States](#), [Parameters](#) and [Constants](#).

For DAEs also specify the mass matrix.

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

The specification of M may depend on parameters and constants.

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

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

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



#### 4.1.7 Initial Conditions

Specify the initial conditions. These may depend on [Parameters](#) or [Constants](#) and must have the same size as  $x$ .

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

#### 4.1.8 Observables

Specify the observables. These may depend on [Parameters](#) and [Constants](#).

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

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

#### 4.1.9 Events

Specifying events is optional. Events are specified in terms of a trigger function, a bolus 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](#).

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

Events may depend on [States](#), [Parameters](#) and [Constants](#) but **not** on [Observables](#)

#### 4.1.10 Standard Deviation

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

Standard deviation for observable data is denoted by `sigma_y`

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

Standard deviation for event data is denoted by `sigma_t`

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

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

#### 4.1.11 Attach to Model Struct

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

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

## 4.2 Model Compilation

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

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

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

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

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

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

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

See also

[amiwrap\(\)](#)

## 4.3 Model Simulation

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

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

### 4.3.1 Integration

Define a time vector:

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

Generate a parameter vector:

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

Generate a constants vector:

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

Integrate:

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

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

Alternatively the integration call also be called via

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

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

### 4.3.2 Forward Sensitivities

Define a time vector:

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

Generate a parameter vector:

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

Generate a constants vector:

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

Set the sensitivity computation to forward sensitivities and Integrate:

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

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

Alternatively the integration call also be called via

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

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

### 4.3.3 Adjoint Sensitivities

Define a time vector:

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

Generate a parameter vector:

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

Set the sensitivity computation to adjoint sensitivities:

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

Define Experimental Data:

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

The NaN values in Sigma\_Y and Sigma\_T will be replaced by the specification in [Standard Deviation](#). Data points with NaN value will be completely ignored.

Generate a constants vector:

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

Integrate:

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

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

#### 4.3.4 Steady State Sensitivities

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

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

Define a final timepoint t:

```
t = 100
```

Generate a parameter vector:

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

Generate a constants vector:

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

Set the sensitivity computation to steady state sensitivities:

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

Integrate:

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

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

## 5 Code Organization

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

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

The fun fields of this object will then be populated by [amimodel::parseModel\(\)](#). The [amimodel::fun](#) field contains all function definitions of type [amifun](#) which are required for model compilation. The set of functions to be considered will depend on the user specification of the model fields [amimodel::adjoint](#) and [amimodel::forward](#) (see [Options](#)) as well as the employed solver (CVODES or IDAS, see [Differential Equation](#)). For all considered functions [amimodel::parseModel\(\)](#) will check their dependencies via [amimodel::checkDeps\(\)](#). These dependencies are a subset

of the user-specified fields of `amimodel::fun` (see [Attach to Model Struct](#)). `amimodel::parseModel()` compares the hashes of all dependencies against the `amimodel::HTable` of possible previous compilations and will only compute necessary symbolic expressions if changes in these fields occurred.

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

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

## 6 Using AMICI-generated code outside Matlab

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

### Generated model files

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

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

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

### Running a simulation

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

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

## Compiling and linking

The complete AMICI API is available through [amici.h](#); this is the only header file that needs to be included. (There are some accessor macro definitions available in `udata_accessors.h`, `rdata_accessors.h` and `edata_accessors.h` which provide shortcuts for accessing struct members of [UserData](#), [ReturnData](#), [ExpData](#), respectively. [amici\\_hdf5.h](#) provides some functions for reading and writing [HDF5](#) files).

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

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

## 7 Hierarchical Index

### 7.1 Class Hierarchy

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

<b>amievent</b>	<b>20</b>
<b>amifun</b>	<b>23</b>
<b>BackwardProblem</b>	<b>70</b>
<b>ExpData</b>	<b>75</b>
<b>ForwardProblem</b>	<b>77</b>
handle	
<b>amidata</b>	<b>15</b>
<b>amimodel</b>	<b>34</b>
<b>amised</b>	<b>66</b>
<b>SBMLode</b>	<b>182</b>
CustomDisplay	
<b>amioption</b>	<b>57</b>
<b>Model</b>	<b>95</b>
<b>modelTest</b>	<b>145</b>
<b>NewtonSolver</b>	<b>146</b>
<b>NewtonSolverDense</b>	<b>148</b>
<b>NewtonSolverIterative</b>	<b>150</b>
<b>NewtonSolverSparse</b>	<b>152</b>

<b>ReturnData</b>	<b>155</b>
<b>ReturnDataMatlab</b>	<b>177</b>
<b>Solver</b>	<b>189</b>
<b>SteadystateProblem</b>	<b>194</b>
sym	
<b>optsym</b>	<b>154</b>
<b>TempData</b>	<b>200</b>
<b>UserData</b>	<b>218</b>

## 8 Class Index

### 8.1 Class List

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

<b>amidata</b>	
<b>AMIDATA</b> 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	<b>15</b>
<b>amievent</b>	
<b>AMIEVENT</b> defines events which later on will be transformed into appropriate C code	<b>20</b>
<b>amifun</b>	
<b>AMIFUN</b> defines functions which later on will be transformed into appropriate C code	<b>23</b>
<b>amimodel</b>	
<b>AMIMODEL</b> carries all model definitions including functions and events	<b>34</b>
<b>amioption</b>	
<b>AMIOPTION</b> provides an option container to pass simulation parameters to the simulation routine	<b>57</b>
<b>amised</b>	
<b>AMISED</b> is a container for SED-ML objects	<b>66</b>
<b>BackwardProblem</b>	
Class to solve backwards problems	<b>70</b>
<b>ExpData</b>	
Struct that carries all information about experimental data	<b>75</b>
<b>ForwardProblem</b>	
Groups all functions for solving the backwards problem. Has only static members	<b>77</b>
<b>Model</b>	
<b>AMICI</b> ODE model. The model does not contain any data, its state should never change	<b>95</b>
<b>modelTest</b>	
<b>MODELTEST</b> Summary of this class goes here Detailed explanation goes here	<b>145</b>
<b>NewtonSolver</b>	<b>146</b>



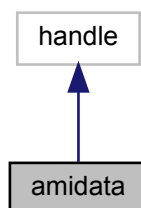
<a href="#">NewtonSolverDense</a>	148
<a href="#">NewtonSolverIterative</a>	150
<a href="#">NewtonSolverSparse</a>	152
<a href="#">optsym</a>	
OPTSYM is an auxiliary class to gain access to the private symbolic property <code>s</code> which is necessary to be able to call <code>symobj::optimize</code> on it	154
<a href="#">ReturnData</a>	
Struct that stores all data which is later returned by the mex function	155
<a href="#">ReturnDataMatlab</a>	
Sets up <a href="#">ReturnData</a> to be returned by the MATLAB mex functions. Memory is allocated using matlab functions	177
<a href="#">SBMLode</a>	
SBMLMODEL provides an intermediate container between the SBML definition and an ami-model object	182
<a href="#">Solver</a>	
<a href="#">Solver</a> class	189
<a href="#">SteadystateProblem</a>	
Solves a steady-state problem using Newton's method and falls back to integration on failure	194
<a href="#">TempData</a>	
Struct that provides temporary storage for different variables	200
<a href="#">UserData</a>	
Struct that stores all user provided data	218

## 9 Class Documentation

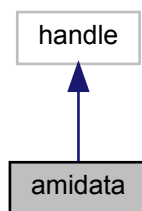
### 9.1 amidata Class Reference

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:



Collaboration diagram for amidata:



### Public Member Functions

- [amidata](#) (matlabtypesubstitute varargin)  
*amidata creates an container for experimental data with specified dimensions amidata.*

### Public Attributes

- matlabtypesubstitute [nt](#) = 0  
*number of timepoints*
- matlabtypesubstitute [ny](#) = 0  
*number of observables*
- matlabtypesubstitute [nz](#) = 0  
*number of event observables*
- matlabtypesubstitute [ne](#) = 0  
*number of events*
- matlabtypesubstitute [nk](#) = 0  
*number of conditions/constants*
- matlabtypesubstitute [t](#) = double.empty("")  
*timepoints of observations*
- matlabtypesubstitute [Y](#) = double.empty("")  
*observations*
- matlabtypesubstitute [Sigma\\_Y](#) = double.empty("")  
*standard deviation of observations*
- matlabtypesubstitute [Z](#) = double.empty("")  
*event observations*
- matlabtypesubstitute [Sigma\\_Z](#) = double.empty("")  
*standard deviation of event observations*
- matlabtypesubstitute [condition](#) = double.empty("")  
*experimental condition*

#### 9.1.1 Detailed Description

Definition at line 17 of file amidata.m.

### 9.1.2 Constructor & Destructor Documentation

#### 9.1.2.1 amidata()

```
amidata (
    matlabtypesubstitute varargin )
```

AMIDATA(amidata) creates a copy of the input container

AMIDATA(struct) tries to creates an amidata container from the input struct. the struct should have the following

fields

t [nt,1] Y [nt,ny] Sigma\_Y [nt,ny] Z [ne,nz] Sigma\_Z [ne,nz] condition [nk,1]

if some fields are missing the function will try to initialise them with NaNs with consistent dimensions

AMIDATA(nt,ny,nz,ne,nk) constructs an empty data container with in the provided dimensions intialised with NaNs

Parameters

<i>varargin</i>	
-----------------	--

Definition at line 122 of file amidata.m.

### 9.1.3 Member Data Documentation

#### 9.1.3.1 nt

nt = 0

**Default:** 0

Note

This property has custom functionality when its value is changed.

Definition at line 31 of file amidata.m.

**9.1.3.2 ny**

```
ny = 0
```

**Default: 0****Note**

This property has custom functionality when its value is changed.

Definition at line 39 of file amidata.m.

**9.1.3.3 nz**

```
nz = 0
```

**Default: 0****Note**

This property has custom functionality when its value is changed.

Definition at line 47 of file amidata.m.

**9.1.3.4 ne**

```
ne = 0
```

**Default: 0****Note**

This property has custom functionality when its value is changed.

Definition at line 55 of file amidata.m.

**9.1.3.5 nk**

```
nk = 0
```

**Default: 0****Note**

This property has custom functionality when its value is changed.

Definition at line 63 of file amidata.m.

#### 9.1.3.6 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.

#### 9.1.3.7 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.

#### 9.1.3.8 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.

#### 9.1.3.9 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.

### 9.1.3.10 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.

### 9.1.3.11 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.

## 9.2 amievent Class Reference

AMIEVENT defines events which later on will be transformed into appropriate C code.

### Public Member Functions

- **amievent** (matlabtypesubstitute **trigger**, matlabtypesubstitute **bolus**, matlabtypesubstitute **z**)  
*amievent constructs an amievent object from the provided input.*
- mlhsInnerSubst< matlabtypesubstitute > **setHflag** (::double **hflag**)  
*gethflag sets the hflag property.*

### Public Attributes

- ::symbolic **trigger** = sym.empty("")  
*the trigger function activates the event on every zero crossing*
- ::symbolic **bolus** = sym.empty("")  
*the bolus function defines the change in states that is applied on every event occurrence*
- ::symbolic **z** = sym.empty("")  
*output function for the event*
- matlabtypesubstitute **hflag** = logical.empty("")  
*flag indicating that a heaviside function is present, this helps to speed up symbolic computations*

### 9.2.1 Detailed Description

Definition at line 17 of file amievent.m.

### 9.2.2 Constructor & Destructor Documentation

#### 9.2.2.1 amievent()

```
amievent (
    matlabtypesubstitute trigger,
    matlabtypesubstitute bolus,
    matlabtypesubstitute z )
```

##### Parameters

<i>trigger</i>	trigger function, the event will be triggered on at all roots of this function
<i>bolus</i>	the bolus that will be added to all states on every occurrence of the event
<i>z</i>	the event output that will be reported on every occurrence of the event

Definition at line 75 of file amievent.m.

Here is the call graph for this function:



### 9.2.3 Member Function Documentation

#### 9.2.3.1 setHflag()

```
mlhsInnerSubst<::amievent> setHflag (
    ::double hflag )
```

##### Parameters

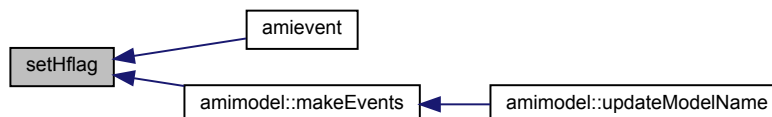
<i>hflag</i>	value for the hflag property
--------------	------------------------------

## 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:



## 9.2.4 Member Data Documentation

### 9.2.4.1 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.

### 9.2.4.2 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.



## 9.2.4.3 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.

## 9.2.4.4 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.

## 9.3 amifun Class Reference

AMIFUN defines functions which later on will be transformed into appropriate C code.

## Public Member Functions

- [amifun](#) (matlabtypesubstitute [funstr](#), matlabtypesubstitute model)
  - amievent constructs an amifun object from the provided input.*
- noret::substitute [printLocalVars](#) (::amimodel model,::fileid fid)
  - printlocalvars prints the C code for the initialisation of local variables into the file specified by fid.*
- 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)
- 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)

*getFArgs* 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 > getFArgs ()`

*getfunargs* populates the *fargstr* property with the argument string of the respective f-function (if applicable). f-function are wrapped implementations of functions which no longer have a model specific name and have solver independent calls.

- `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

## 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)*
- `::char fargstr = char.empty("")`  
*argument string (solver unspecific)*
- `::cell deps = cell.empty("")`  
*dependencies on other functions*
- `matlabtypesubstitute nvecs = cell.empty("")`  
*nvec dependencies*
- `matlabtypesubstituteensiflag = logical.empty("")`  
*indicates whether the function is a sensitivity or derivative with respect to parameters*

### 9.3.1 Detailed Description

Definition at line 17 of file `amifun.m`.

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 `amifun()`

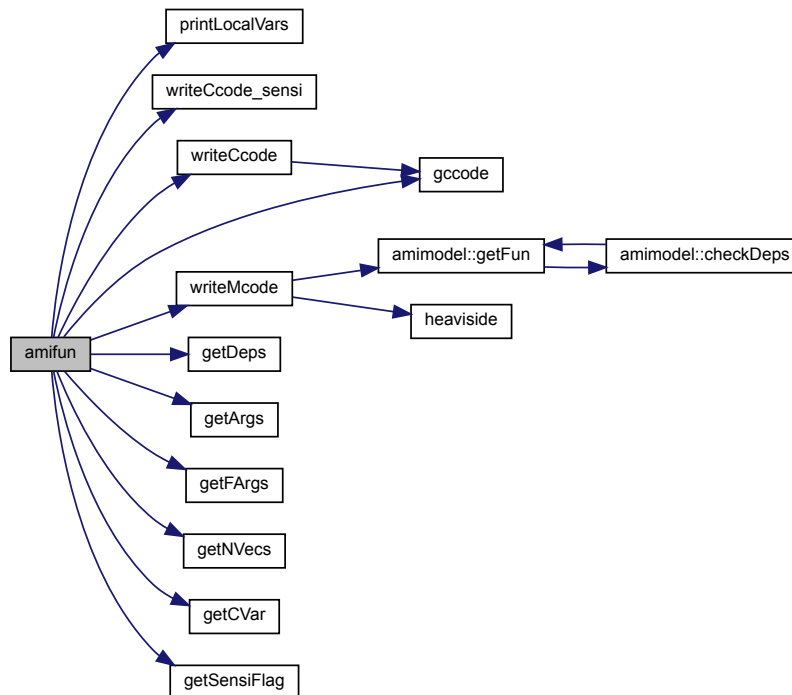
```
amifun (
    matlabtypesubstitute funstr,
    matlabtypesubstitute model )
```

## 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 119 of file amifun.m.

Here is the call graph for this function:



## 9.3.3 Member Function Documentation

## 9.3.3.1 printLocalVars()

```

noret::substitute printLocalVars (
    ::amimodel model,
    ::fileid fid )

```

## Parameters

<i>model</i>	this struct must contain all necessary symbolic definitions
<i>fid</i>	file id in which the final expression is written

**Return values**

<i>fid</i>	Nothing
------------	---------

Definition at line 18 of file printLocalVars.m.

Here is the caller graph for this function:

**9.3.3.2 writeCcode\_sensi()**

```

noret::substitute writeCcode_sensi (
    ::amimodel model,
    ::fileid fid )
  
```

**Parameters**

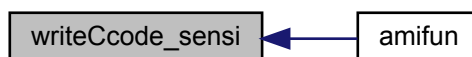
<i>model</i>	model defintion object
<i>fid</i>	file id in which the final expression is written

**Return values**

<i>fid</i>	void
------------	------

Definition at line 18 of file writeCcode\_sensi.m.

Here is the caller graph for this function:



### 9.3.3.3 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:



Here is the caller graph for this function:



### 9.3.3.4 writeMcode()

```
noret::substitute writeMcode (  
    ::amimodel model )
```

#### Parameters

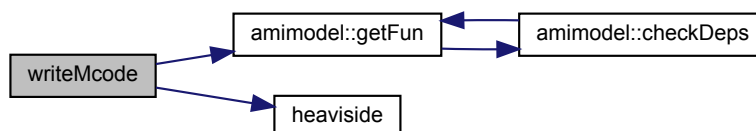
<i>model</i>	model defintion object
--------------	------------------------

## Return values

<i>model</i>	void
--------------	------

Definition at line 18 of file writeMcode.m.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.3.3.5 gccode()

```

mlhsInnerSubst<::amifun > gccode (
    ::amimodel model,
    ::fileid fid )

```

## Parameters

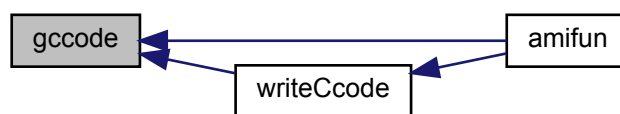
<i>model</i>	model definition object
<i>fid</i>	file id in which the expression should be written

## Return values

<i>this</i>	function definition object
-------------	----------------------------

Definition at line 18 of file gccode.m.

Here is the caller graph for this function:



#### 9.3.3.6 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`.

Here is the caller graph for this function:



#### 9.3.3.7 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.

Here is the caller graph for this function:

**9.3.3.8 getFArgs()**

```
mlhsInnerSubst<::amifun > getFArgs ( )
```

**Return values**

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getFArgs.m.

Here is the caller graph for this function:

**9.3.3.9 getNVecs()**

```
mlhsInnerSubst<::amifun > getNVecs ( )
```



## Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getNVecs.m.

Here is the caller graph for this function:



## 9.3.3.10 getCVar()

```
mlhsInnerSubst<::amifun > getCVar ( )
```

## Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getCVar.m.

Here is the caller graph for this function:



## 9.3.3.11 getSensiFlag()

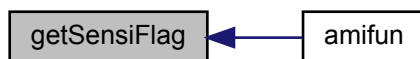
```
mlhsInnerSubst<::amifun > getSensiFlag ( )
```

## Return values

<i>this</i>	updated function definition object
-------------	------------------------------------

Definition at line 18 of file getSensiFlag.m.

Here is the caller graph for this function:



### 9.3.4 Member Data Documentation

#### 9.3.4.1 sym

```
sym = sym("[]")
```

**Default:** sym("[]")

Definition at line 27 of file amifun.m.

#### 9.3.4.2 sym\_noopt

```
sym_noopt = sym("[]")
```

**Default:** sym("[]")

Definition at line 35 of file amifun.m.

#### 9.3.4.3 strsym

```
strsym = sym("[]")
```

**Default:** sym("[]")

Definition at line 43 of file amifun.m.

#### 9.3.4.4 strsym\_old

```
strsym_old = sym("[]")
```

**Default:** sym("[]")

Definition at line 51 of file amifun.m.

#### 9.3.4.5 funstr

```
funstr = char.empty("")
```

**Default:** char.empty("")

Definition at line 59 of file amifun.m.

#### 9.3.4.6 cvar

```
cvar = char.empty("")
```

**Default:** char.empty("")

Definition at line 67 of file amifun.m.

#### 9.3.4.7 argstr

```
argstr = char.empty("")
```

**Default:** char.empty("")

Definition at line 75 of file amifun.m.

#### 9.3.4.8 fargstr

```
fargstr = char.empty("")
```

**Default:** char.empty("")

Definition at line 83 of file amifun.m.

#### 9.3.4.9 deps

```
deps = cell.empty("")
```

**Default:** cell.empty("")

Definition at line 91 of file amifun.m.

#### 9.3.4.10 nvecs

```
nvecs = cell.empty("")
```

**Default:** cell.empty("")

Definition at line 99 of file amifun.m.

#### 9.3.4.11 sensiflag

```
sensiflag = logical.empty("")
```

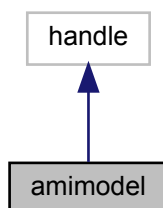
**Default:** logical.empty("")

Definition at line 107 of file amifun.m.

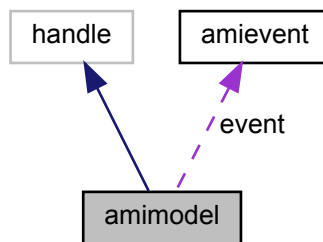
## 9.4 amimodel Class Reference

AMIMODEL carries all model definitions including functions and events.

Inheritance diagram for amimodel:



Collaboration 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 updateModelName (matlabtypesubstitute modelName)`  
*updateModelName updates the modelName*
- `noret::substitute parseModel ()`  
*parseModel parses the model definition and computes all necessary symbolic expressions.*
- `noret::substitute generateC ()`  
*generateC generates the c files which will be used in the compilation.*
- `noret::substitute compileC ()`  
*compileC compiles the mex simulation file*
- `noret::substitute generateM (::amimodel amimodelo2)`  
*generateM generates the matlab wrapper for the compiled C files.*
- `noret::substitute getFun (::struct HTable,::string funstr)`  
*getFun generates symbolic expressions for the requested function.*
- `noret::substitute makeEvents ()`  
*makeEvents extracts discontinuities from the model right hand side and converts them into events*
- `noret::substitute makeSyms ()`  
*makeSyms extracts symbolic definition from the user provided model and checks them for consistency*
- `mlhsInnerSubst< matlabtypesubstitute > checkDeps (::struct HTable,::cell deps)`  
*checkDeps checks the dependencies of functions and populates sym fields if necessary*
- `mlhsInnerSubst< matlabtypesubstitute > loadOldHashes ()`  
*loadOldHashes loads information from a previous compilation of the model.*
- `mlhsInnerSubst< 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.*
- `mlhsInnerSubst< 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 [wrap\\_path](#), matlabtypesubstitute [recompile](#), matlabtypesubstitute [coptim](#), matlabtypesubstitute [debug](#), matlabtypesubstitute [funcs](#), matlabtypesubstitute [cfun](#), matlabtypesubstitute [adjoint](#))

*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\(\)](#).*

## 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*
- ::bool [steadystate](#) = true  
*flag indicating whether steady state sensitivities should be enabled*
- ::double [t0](#) = 0  
*default initial time*
- ::string [wtype](#) = char.empty("")  
*type of wrapper (cvodes/idas)*
- ::int [nx](#) = double.empty("")  
*number of states*
- ::int [nxtrue](#) = double.empty("")  
*number of original states for second order sensitivities*
- ::int [ny](#) = double.empty("")  
*number of observables*
- ::int [nytrue](#) = double.empty("")  
*number of original observables for second order sensitivities*
- ::int [np](#) = double.empty("")  
*number of parameters*
- ::int [nk](#) = double.empty("")  
*number of constants*
- ::int [ng](#) = double.empty("")  
*number of objective functions*
- ::int [nevent](#) = double.empty("")  
*number of events*
- ::int [nz](#) = double.empty("")  
*number of event outputs*

- `::int nztrue = double.empty("")`  
*number of original event outputs for second order sensitivities*
- `::*int id = double.empty("")`  
*flag for DAEs*
- `::int ubw = double.empty("")`  
*upper Jacobian bandwidth*
- `::int lbw = double.empty("")`  
*lower Jacobian bandwidth*
- `::int 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 mfun = cell.empty("")`  
*cell array of matlab functions to be compiled*
- `::string coptim = "-O3"`  
*optimisation flag for compilation*
- `::string param = "lin"`  
*default parametrisation*
- `matlabtypesubstitute wrap_path = char.empty("")`  
*path to wrapper*
- `matlabtypesubstitute recompile = false`  
*flag to enforce recompilation of the model*
- `matlabtypesubstitute cfun = struct.empty("")`  
*storage for flags determining recompilation of individual functions*
- `matlabtypesubstitute 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)*
- `matlabtypesubstitute z2event = double.empty("")`  
*vector that maps outputs to events*
- `matlabtypesubstitute splineflag = false`  
*flag indicating whether the model contains spline functions*
- `matlabtypesubstitute minflag = false`  
*flag indicating whether the model contains min functions*
- `matlabtypesubstitute 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  $w$ ,  $dwdp$*

#### 9.4.1 Detailed Description

Definition at line 17 of file amimodel.m.

#### 9.4.2 Constructor & Destructor Documentation

##### 9.4.2.1 amimodel()

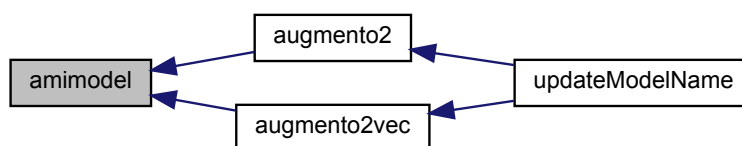
```
amimodel (
    ::string symfun,
    ::string modelname )
```

##### 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 526 of file amimodel.m.

Here is the caller graph for this function:



#### 9.4.3 Member Function Documentation

##### 9.4.3.1 updateRHS()

```
noret::substitute updateRHS (
    matlabtypesubstitute xdot )
```



## Parameters

<i>xdot</i>	new right hand side of the differential equation
-------------	--

Definition at line 623 of file amimodel.m.

## 9.4.3.2 updateModelName()

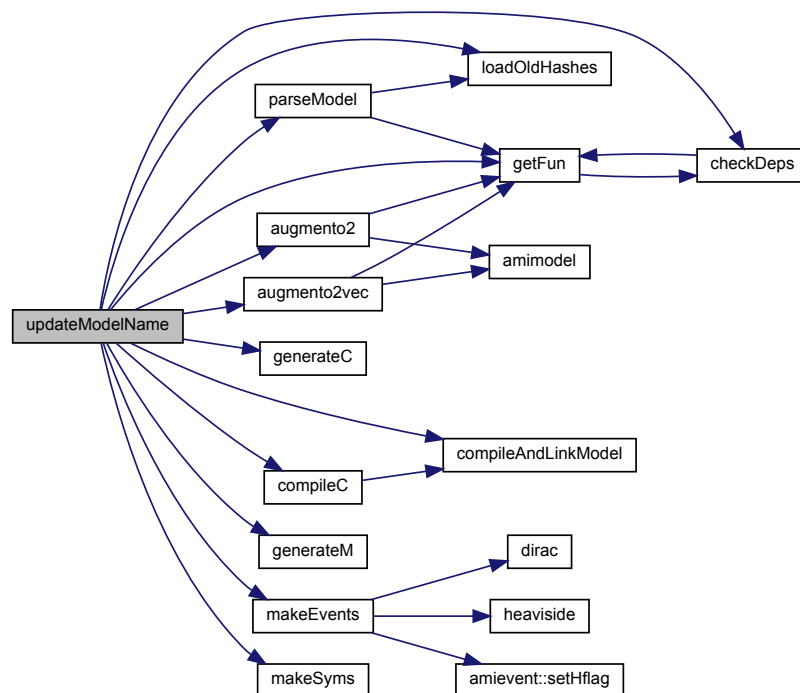
```
noret::substitute updateModelName (
    matlabtypesubstitute modelname )
```

## Parameters

<i>modelname</i>	new modelname
------------------	---------------

Definition at line 636 of file amimodel.m.

Here is the call graph for this function:



## 9.4.3.3 generateC()

```
noret::substitute generateC ( )
```

**Return values**

<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file generateC.m.

Here is the caller graph for this function:

**9.4.3.4 compileC()**

```
noret::substitute compileC ( )
```

**Return values**

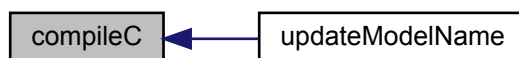
<i>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file compileC.m.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.4.3.5 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>this</i>	model definition object
-------------	-------------------------

Definition at line 18 of file generateM.m.

Here is the caller graph for this function:



## 9.4.3.6 getFun()

```
noret::substitute getFun (
    ::struct HTable,
    ::string funstr )
```

## Parameters

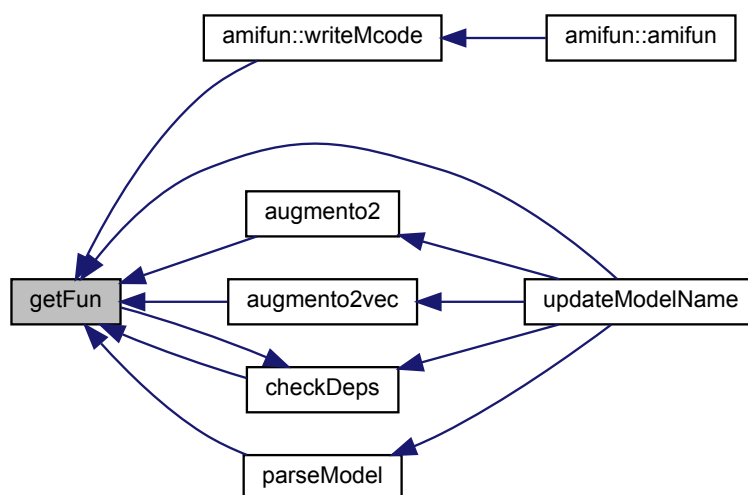
<i>HTable</i>	struct with hashes of symbolic definition from the previous compilation
<i>funstr</i>	function for which symbolic expressions should be computed

Definition at line 18 of file getFun.m.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.4.3.7 checkDeps()

```

mlhsInnerSubst<::bool> checkDeps (
    ::struct HTable,
    ::cell deps )
  
```

##### Parameters

<i>HTable</i>	struct with reference hashes of functions in its fields
<i>deps</i>	cell array with containing a list of dependencies

##### Return values

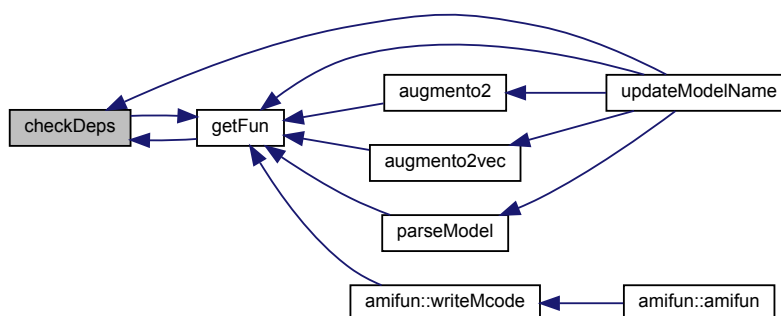
<i>cflag</i>	boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable
--------------	---

Definition at line 18 of file checkDeps.m.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.4.3.8 loadOldHashes()

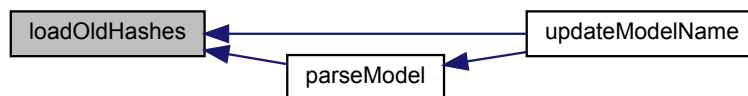
```
mlhsInnerSubst<::struct > loadOldHashes ( )
```

**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:



#### 9.4.3.9 augmento2()

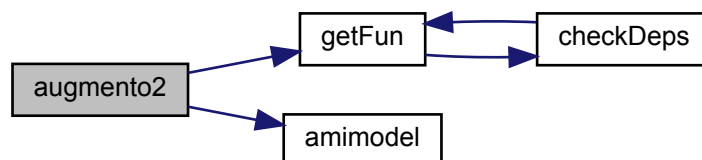
```
mlhsInnerSubst< matlabtypesubstitute > augmento2 ( )
```

##### 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:



Here is the caller graph for this function:



## 9.4.3.10 augmento2vec()

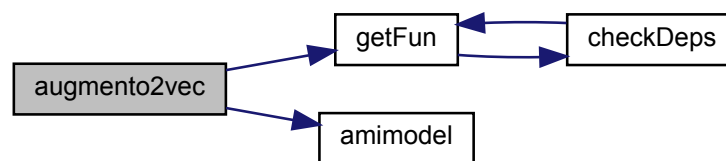
```
mlhsInnerSubst< matlabtypesubstitute > augmento2vec ( )
```

## Return values

<i>this</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:



Here is the caller graph for this function:



## 9.4.3.11 compileAndLinkModel()

```
noret::substitute compileAndLinkModel (
    matlabtypesubstitute modelname,
    matlabtypesubstitute wrap_path,
    matlabtypesubstitute recompile,
    matlabtypesubstitute coptim,
    matlabtypesubstitute debug,
    matlabtypesubstitute funs,
    matlabtypesubstitute cfun,
    matlabtypesubstitute adjoint ) [static]
```

## Parameters

<i>modelName</i>	name of the model as specified for <a href="#">amiwrap()</a> wrap_path AMICI path recompile flag indicating whether all source files should be recompiled coptim optimization flags debug enable debugging? funs array with names of the model functions, will be guessed from source files if left empty cfun struct indicating which files should be recompiled adjoint flag indicating whether adjoint sensitivities are enabled
------------------	---

Definition at line 18 of file compileAndLinkModel.m.

Here is the caller graph for this function:



#### 9.4.4 Member Data Documentation

##### 9.4.4.1 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.

##### 9.4.4.2 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.



#### 9.4.4.3 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.

#### 9.4.4.4 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.

#### 9.4.4.5 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.

#### 9.4.4.6 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.

#### 9.4.4.7 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.

#### 9.4.4.8 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.

#### 9.4.4.9 steadystate

```
steadystate = true
```

##### Note

This property has non-standard access specifiers: SetAccess = Private, GetAccess = Public

[Matlab documentation of property attributes.](#)

**Default:** true

Definition at line 116 of file amimodel.m.

#### 9.4.4.10 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 127 of file amimodel.m.

#### 9.4.4.11 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 138 of file amimodel.m.

#### 9.4.4.12 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 149 of file amimodel.m.

#### 9.4.4.13 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 160 of file amimodel.m.

#### 9.4.4.14 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 171 of file amimodel.m.

#### 9.4.4.15 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 182 of file amimodel.m.

#### 9.4.4.16 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 193 of file amimodel.m.

#### 9.4.4.17 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 204 of file amimodel.m.

#### 9.4.4.18 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 215 of file amimodel.m.

#### 9.4.4.19 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 226 of file amimodel.m.

#### 9.4.4.20 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 237 of file amimodel.m.

#### 9.4.4.21 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 248 of file amimodel.m.

#### 9.4.4.22 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 259 of file amimodel.m.

#### 9.4.4.23 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 270 of file amimodel.m.

#### 9.4.4.24 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 281 of file amimodel.m.

#### 9.4.4.25 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 292 of file amimodel.m.

#### 9.4.4.26 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 303 of file amimodel.m.

#### 9.4.4.27 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 314 of file amimodel.m.

#### 9.4.4.28 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 325 of file amimodel.m.

#### 9.4.4.29 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 336 of file amimodel.m.

#### 9.4.4.30 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 347 of file amimodel.m.

#### 9.4.4.31 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 358 of file amimodel.m.

#### 9.4.4.32 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 369 of file amimodel.m.

#### 9.4.4.33 mfun

```
mfuns = 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 380 of file amimodel.m.

#### 9.4.4.34 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 391 of file amimodel.m.



#### 9.4.4.35 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 402 of file amimodel.m.

#### 9.4.4.36 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 413 of file amimodel.m.

#### 9.4.4.37 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 424 of file amimodel.m.

#### 9.4.4.38 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 435 of file amimodel.m.

#### 9.4.4.39 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 447 of file amimodel.m.

#### 9.4.4.40 z2event

```
z2event = double.empty("")
```

**Default:** double.empty("")

Definition at line 466 of file amimodel.m.

#### 9.4.4.41 splineflag

```
splineflag = false
```

**Default:** false

Definition at line 474 of file amimodel.m.

#### 9.4.4.42 minflag

```
minflag = false
```

**Default:** false

Definition at line 482 of file amimodel.m.

#### 9.4.4.43 maxflag

```
maxflag = false
```

**Default:** false

Definition at line 490 of file amimodel.m.

#### 9.4.4.44 nw

`nw = 0`

**Default:** 0

Definition at line 498 of file amimodel.m.

#### 9.4.4.45 ndwdx

`ndwdx = 0`

**Default:** 0

Definition at line 507 of file amimodel.m.

#### 9.4.4.46 ndwdp

`ndwdp = 0`

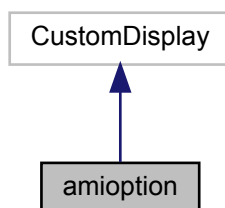
**Default:** 0

Definition at line 515 of file amimodel.m.

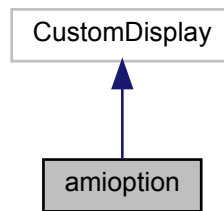
## 9.5 amioption Class Reference

AMIOPTION provides an option container to pass simulation parameters to the simulation routine.

Inheritance diagram for amioption:



Collaboration 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 tolerace*
- matlabtypesubstitute `rtol` = 1e-8  
*relative integration tolerace*
- matlabtypesubstitute `maxsteps` = 1e4  
*maximum number of integration steps*
- matlabtypesubstitute `sens_ind` = double.empty("")  
*index of parameters for which the sensitivities are computed*
- matlabtypesubstitute `qpositivex` = double.empty("")  
*index of states for which positivity should be enforced*
- matlabtypesubstitute `pbar` = double.empty("")  
*scaling of error tolerances for sensitivity equations*
- 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*
- matlabtypesubstitute `interpType` = 1  
*interpolation type*
- matlabtypesubstitute `lmmB` = 2  
*linear multistep method (backwards)*
- matlabtypesubstitute `iterB` = 2

- iteration method for linear multistep (backwards).*
- matlabtypesubstitute `ism` = 1  
*forward sensitivity mode*
- matlabtypesubstitute `sensi_meth` = 1  
*sensitivity method*
- matlabtypesubstitute `sensi` = 0  
*sensitivity order*
- matlabtypesubstitute `nmaxevent` = 10  
*number of reported events*
- matlabtypesubstitute `ordering` = 0  
*reordering of states*
- matlabtypesubstitute `ss` = 0  
*steady state sensitivity flag*
- matlabtypesubstitute `x0` = double.empty("")  
*custom initial state*
- matlabtypesubstitute `sx0` = double.empty("")  
*custom initial sensitivity*
- matlabtypesubstitute `newton_precon` = 1  
*newton solver: preconditioning method (0 = none, 1 = diagonal, 2 = incomplete LU)*
- matlabtypesubstitute `newton_maxsteps` = 40  
*newton solver: maximum newton steps*
- matlabtypesubstitute `newton_maxlinsteps` = 100  
*newton solver: maximum linear steps*
- matlabtypesubstitute `newton_preeq` = false  
*preequilibration of system via newton solver*
- matlabtypesubstitute `z2event` = double.empty("")  
*mapping of event outputs to events*
- matlabtypesubstitute `pscale` = ""  
*parameter scaling Valid options are "log", "log10" and "lin" for log, log10 or unscaled parameters p use "" for default as specified in the model (fallback: lin)*
- matlabtypesubstitute `id` = double.empty("")  
*flag for DAE variables*

### 9.5.1 Detailed Description

Definition at line 17 of file amioption.m.

### 9.5.2 Constructor & Destructor Documentation

#### 9.5.2.1 amioption()

```
amioption (
    matlabtypesubstitute varargin )
```

OPTS = amioption(PARAM, VAL, ...) creates a set of options with the named parameters altered with the specified values.

OPTS = amioption(OLDOPTS, PARAM, VAL, ...) creates a copy of OLDOPTS with the named parameters altered with the specified value

Note: to see the parameters, check the documentation page for

**Parameters**

<i>varargin</i>	input to construct amioption object, see function function description
-----------------	--

Definition at line 274 of file amioption.m.

**9.5.3 Member Data Documentation****9.5.3.1 atol**

```
atol = 1e-16
```

**Default:** 1e-16

Definition at line 28 of file amioption.m.

**9.5.3.2 rtol**

```
rtol = 1e-8
```

**Default:** 1e-8

Definition at line 36 of file amioption.m.

**9.5.3.3 maxsteps**

```
maxsteps = 1e4
```

**Default:** 1e4

Definition at line 44 of file amioption.m.

**9.5.3.4 sens\_ind**

```
sens_ind = double.empty("")
```

**Default:** double.empty("")

Definition at line 52 of file amioption.m.

#### 9.5.3.5 qpositivex

```
qpositivex = double.empty("")
```

**Default:** double.empty("")

Definition at line 60 of file amioption.m.

#### 9.5.3.6 pbar

```
pbar = double.empty("")
```

**Default:** double.empty("")

Definition at line 68 of file amioption.m.

#### 9.5.3.7 tstart

```
tstart = 0
```

**Default:** 0

Definition at line 76 of file amioption.m.

#### 9.5.3.8 lmm

```
lmm = 2
```

**Default:** 2

Definition at line 84 of file amioption.m.

#### 9.5.3.9 iter

```
iter = 2
```

**Default:** 2

Definition at line 92 of file amioption.m.

**9.5.3.10 linsol**

```
linsol = 9
```

**Default:** 9

Definition at line 100 of file amioption.m.

**9.5.3.11 stldet**

```
stldet = true
```

**Default:** true

Definition at line 108 of file amioption.m.

**9.5.3.12 interpType**

```
interpType = 1
```

**Default:** 1

Definition at line 116 of file amioption.m.

**9.5.3.13 lmmB**

```
lmmB = 2
```

**Default:** 2

Definition at line 124 of file amioption.m.

**9.5.3.14 iterB**

```
iterB = 2
```

**Default:** 2

Definition at line 132 of file amioption.m.



#### 9.5.3.15 ism

```
ism = 1
```

**Default:** 1

Definition at line 140 of file amioption.m.

#### 9.5.3.16 sensi\_meth

```
sensi_meth = 1
```

**Default:** 1

##### Note

This property has custom functionality when its value is changed.

Definition at line 148 of file amioption.m.

#### 9.5.3.17 sensi

```
sensi = 0
```

**Default:** 0

##### Note

This property has custom functionality when its value is changed.

Definition at line 156 of file amioption.m.

#### 9.5.3.18 nmaxevent

```
nmaxevent = 10
```

**Default:** 10

Definition at line 164 of file amioption.m.

#### 9.5.3.19 ordering

```
ordering = 0
```

**Default:** 0

Definition at line 172 of file amioption.m.

#### 9.5.3.20 ss

```
ss = 0
```

**Default:** 0

Definition at line 180 of file amioption.m.

#### 9.5.3.21 x0

```
x0 = double.empty("")
```

**Default:** double.empty("")

Definition at line 188 of file amioption.m.

#### 9.5.3.22 sx0

```
sx0 = double.empty("")
```

**Default:** double.empty("")

Definition at line 196 of file amioption.m.

#### 9.5.3.23 newton\_precon

```
newton_precon = 1
```

**Default:** 1

#### Note

This property has custom functionality when its value is changed.

Definition at line 204 of file amioption.m.

#### 9.5.3.24 newton\_maxsteps

```
newton_maxsteps = 40
```

**Default:** 40

##### Note

This property has custom functionality when its value is changed.

Definition at line 213 of file amioption.m.

#### 9.5.3.25 newton\_maxlinsteps

```
newton_maxlinsteps = 100
```

**Default:** 100

##### Note

This property has custom functionality when its value is changed.

Definition at line 221 of file amioption.m.

#### 9.5.3.26 newton\_preeq

```
newton_preeq = false
```

**Default:** false

##### Note

This property has custom functionality when its value is changed.

Definition at line 229 of file amioption.m.

#### 9.5.3.27 z2event

```
z2event = double.empty("")
```

**Default:** double.empty("")

Definition at line 237 of file amioption.m.

### 9.5.3.28 pscale

```
pscale = ""
```

**Default:** ""

#### Note

This property has custom functionality when its value is changed.

Definition at line 245 of file amioption.m.

### 9.5.3.29 id

```
id = double.empty("")
```

#### Note

This property has the MATLAB attribute `Hidden` set to `true`.

[Matlab documentation of property attributes.](#)

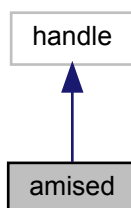
**Default:** `double.empty("")`

Definition at line 260 of file amioption.m.

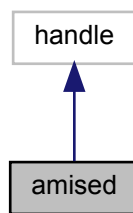
## 9.6 amised Class Reference

AMISED is a container for SED-ML objects.

Inheritance diagram for amised:



Collaboration diagram for amised:



### Public Member Functions

- `amised` (matlabtypesubstitute sedname)  
*amised reads in an SEDML document using the JAVA binding of 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*

#### 9.6.1 Detailed Description

Definition at line 17 of file `amised.m`.

#### 9.6.2 Constructor & Destructor Documentation

##### 9.6.2.1 `amised()`

```

amised (
    matlabtypesubstitute sedname )
  
```

**Parameters**

<i>sedname</i>	name/path of the SEDML document
----------------	---------------------------------

Definition at line 112 of file amised.m.

Here is the call graph for this function:

**9.6.3 Member Data Documentation****9.6.3.1 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.

**9.6.3.2 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.

### 9.6.3.3 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.

### 9.6.3.4 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.

### 9.6.3.5 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.

### 9.6.3.6 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.

### 9.6.3.7 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.

## 9.7 BackwardProblem Class Reference

class to solve backwards problems.

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

### Static Public Member Functions

- static int [workBackwardProblem](#) ([UserData](#) \*udata, [TempData](#) \*tdata, [ReturnData](#) \*rdata, [Model](#) \*model)
- static int [handleEventB](#) (int iroot, [TempData](#) \*tdata, [Model](#) \*model)
- static int [handleDataPointB](#) (int it, [ReturnData](#) \*rdata, [TempData](#) \*tdata, [Solver](#) \*solver, [Model](#) \*model)
- static int [updateHeavisideB](#) (int iroot, [TempData](#) \*tdata, int ne)
- static realtype [getNext](#) (realtype \*troot, int iroot, realtype \*tdata, int it, [Model](#) \*model)

### 9.7.1 Detailed Description

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

Definition at line 19 of file backwardproblem.h.

### 9.7.2 Member Function Documentation

#### 9.7.2.1 workBackwardProblem()

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

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



## Parameters

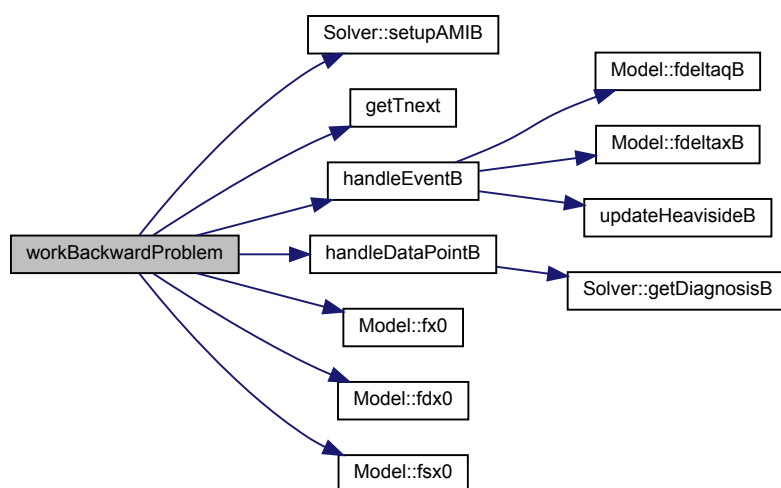
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

## Returns

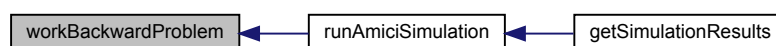
int status flag

Definition at line 10 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.7.2.2 handleEventB()

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

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

#### Parameters

out	<i>iroot</i>	index of event <b>Type:</b> int
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> TempData
in	<i>model</i>	pointer to model specification object <b>Type:</b> Model

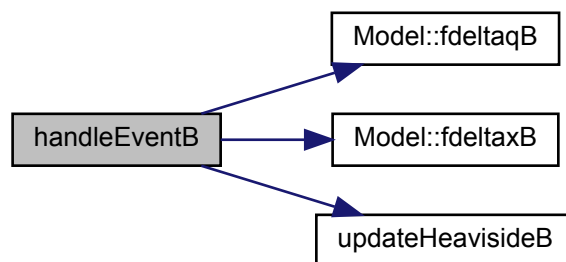
#### Returns

status flag indicating success of execution

**Type:** int

Definition at line 174 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.7.2.3 handleDataPointB()

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

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

## Parameters

in	<i>it</i>	index of data point <b>Type:</b> int
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>solver</i>	pointer to solver object <b>Type:</b> <a href="#">Solver</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

## Returns

status flag indicating success of execution  
**Type:** int

Definition at line 246 of file backwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.7.2.4 updateHeavisideB()

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

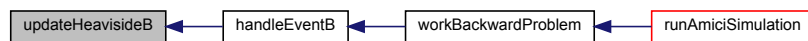
updateHeavisideB updates the heaviside variables h on event occurrences for the backward problem

##### Parameters

in	<i>iroot</i>	discontinuity occurrence index <b>Type:</b> int
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> TempData

Definition at line 277 of file backwardproblem.cpp.

Here is the caller graph for this function:



#### 9.7.2.5 getTnext()

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

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

##### Parameters

in	<i>troot</i>	timepoint of next event <b>Type:</b> realtype
in	<i>iroot</i>	index of next event <b>Type:</b> int
in	<i>tdata</i>	timepoint of next data point <b>Type:</b> realtype
in	<i>it</i>	index of next data point <b>Type:</b> int
in	<i>model</i>	pointer to model specification object <b>Type:</b> Model

## Returns

tnext next timepoint

**Type:** realtype

Definition at line 301 of file backwardproblem.cpp.

Here is the caller graph for this function:



## 9.8 ExpData Class Reference

struct that carries all information about experimental data

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

### Public Member Functions

- [ExpData](#) ()  
*Default constructor.*
- [ExpData](#) (const [UserData](#) \*udata, [Model](#) \*model)
- void [setDefault](#) ()

### Public Attributes

- double \* [my](#)
- double \* [sigmay](#)
- double \* [mz](#)
- double \* [mrz](#)
- double \* [sigmaz](#)

#### 9.8.1 Detailed Description

Definition at line 8 of file edata.h.

#### 9.8.2 Constructor & Destructor Documentation

### 9.8.2.1 ExpData()

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

initialization with `UserData` and model

Definition at line 9 of file edata.cpp.

Here is the call graph for this function:



## 9.8.3 Member Function Documentation

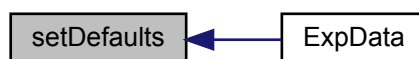
### 9.8.3.1 setDefaults()

```
void setDefaults ( )
```

initialization with default values

Definition at line 20 of file edata.cpp.

Here is the caller graph for this function:



## 9.8.4 Member Data Documentation

#### 9.8.4.1 my

`double* my`

observed data

Definition at line 23 of file edata.h.

#### 9.8.4.2 sigmay

`double* sigmay`

standard deviation of observed data

Definition at line 25 of file edata.h.

#### 9.8.4.3 mz

`double* mz`

observed events

Definition at line 28 of file edata.h.

#### 9.8.4.4 mrz

`double* mrz`

observed roots

Definition at line 30 of file edata.h.

#### 9.8.4.5 sigmaz

`double* sigmaz`

standard deviation of observed events/roots

Definition at line 32 of file edata.h.

## 9.9 ForwardProblem Class Reference

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

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

## Static Public Member Functions

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

### 9.9.1 Detailed Description

Definition at line 18 of file forwardproblem.h.

### 9.9.2 Member Function Documentation

#### 9.9.2.1 [workForwardProblem\(\)](#)

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

[workForwardProblem](#) solves the forward problem. if forward sensitivities are enabled this will also compute sensitivities

#### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
out	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>



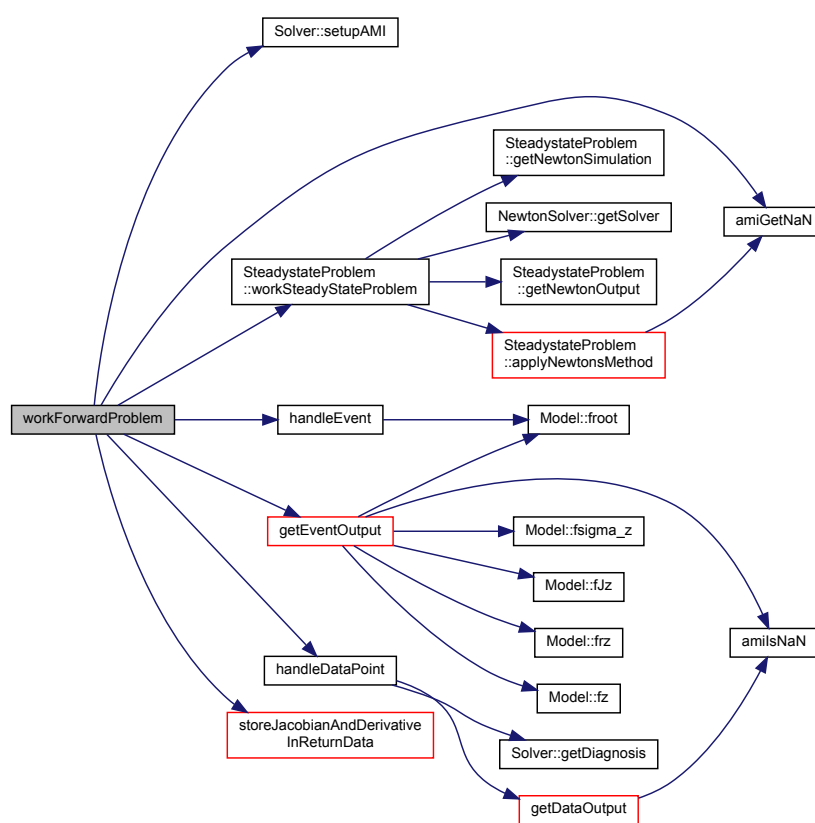
## Returns

int status flag indicating success of execution

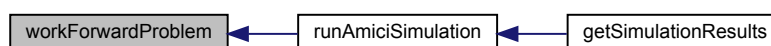
**Type:** int

Definition at line 11 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.9.2.2 `handleEvent()`

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

`handleEvent` executes everything necessary for the handling of events

#### Parameters

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

#### Returns

status flag indicating success of execution  
**Type:** *int*

Definition at line 123 of file `forwardproblem.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.9.2.3 storeJacobianAndDerivativeInReturnData()

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

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

#### Parameters

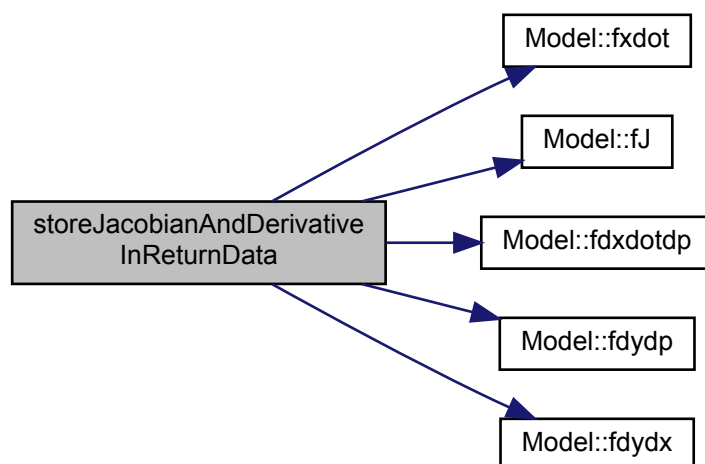
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

#### Returns

void

Definition at line 321 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.9.2.4 getEventOutput()

```

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

`getEventOutput` extracts output information for events

##### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

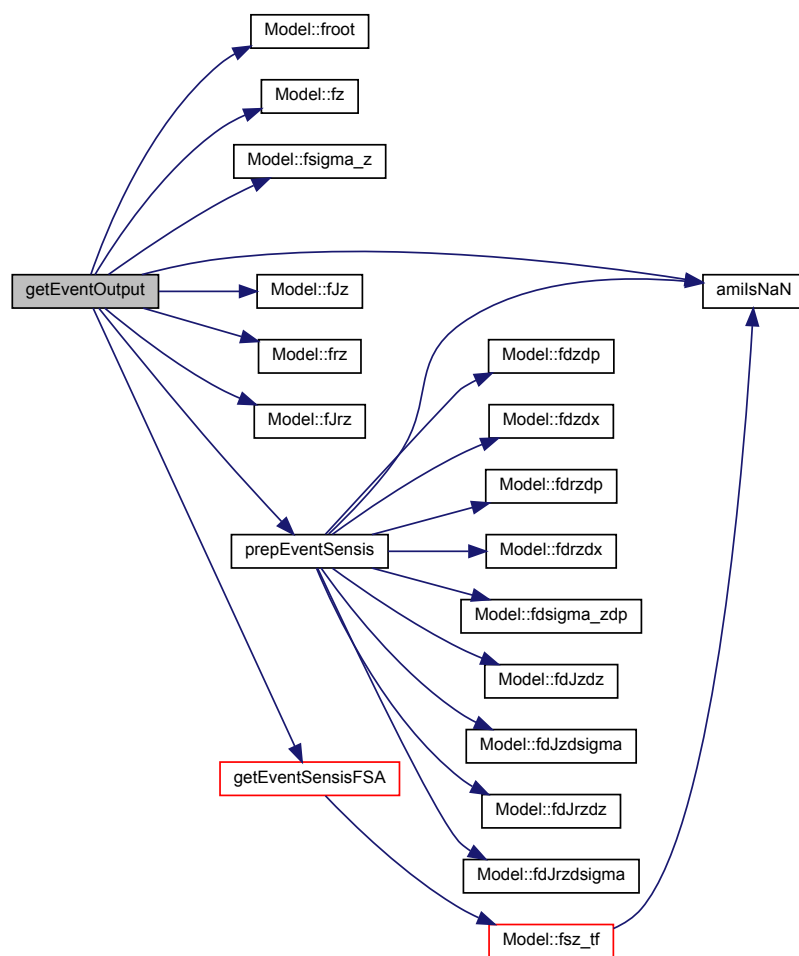
## Returns

status flag indicating success of execution

**Type:** int

Definition at line 396 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.9.2.5 `prepEventSensis()`

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

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

#### Parameters

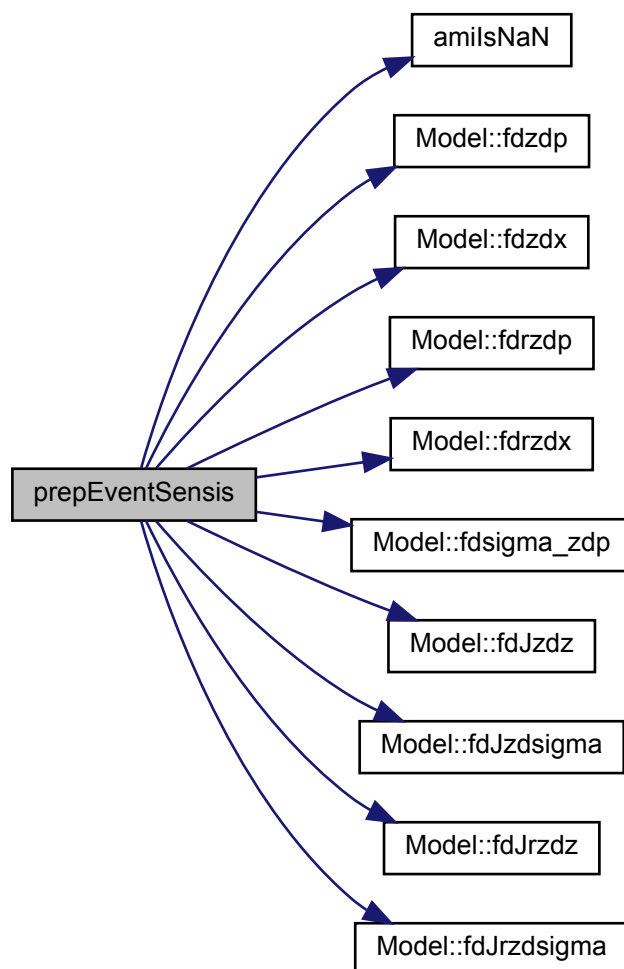
in	<i>ie</i>	index of current event <b>Type:</b> int
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

#### Returns

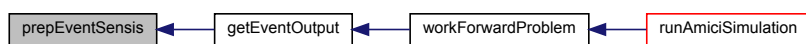
status flag indicating success of execution  
**Type:** int

Definition at line 495 of file `forwardproblem.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.9.2.6 getEventSensisFSA()

```

int getEventSensisFSA (
    int ie,

```

```

ReturnData * rdata,
const ExpData * edata,
TempData * tdata,
Model * model ) [static]

```

getEventSensisFSA extracts event information for forward sensitivity analysis

#### Parameters

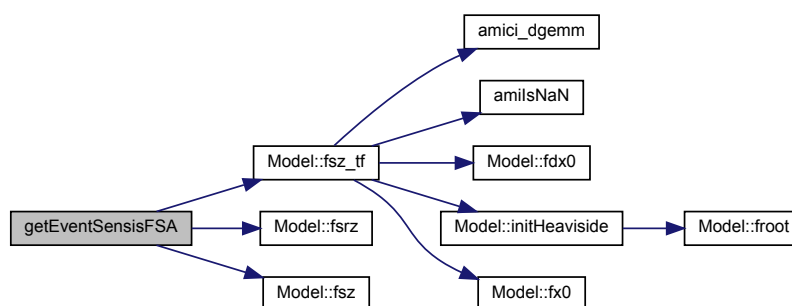
in	<i>ie</i>	index of event type <b>Type:</b> int
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

#### Returns

status flag indicating success of execution  
**Type:** int

Definition at line 608 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:





## 9.9.2.7 handleDataPoint()

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

handleDataPoint executes everything necessary for the handling of data points

## Parameters

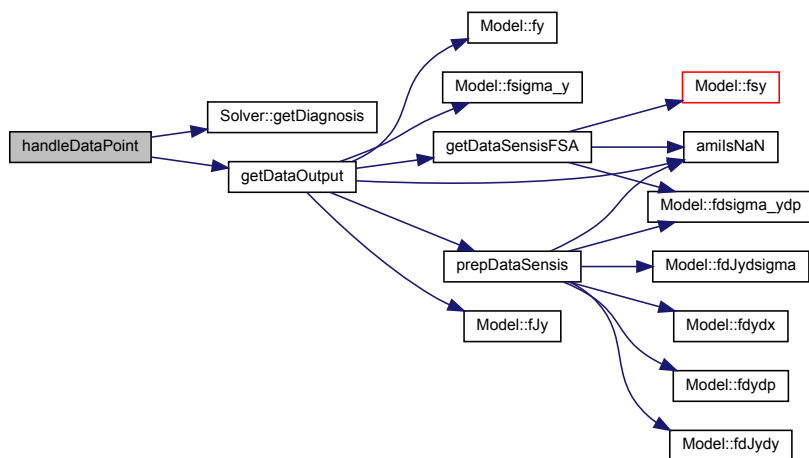
in	<i>it</i>	index of data point <b>Type:</b> int
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>solver</i>	pointer to solver object <b>Type:</b> <a href="#">Solver</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

## Returns

status flag indicating success of execution  
**Type:** int

Definition at line 651 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.9.2.8 getDataOutput()

```

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

getDataOutput extracts output information for data-points

#### Parameters

in	<i>it</i>	index of current timepoint <b>Type:</b> int
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>

## Parameters

out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>solver</i>	pointer to solver object <b>Type:</b> <a href="#">Solver</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

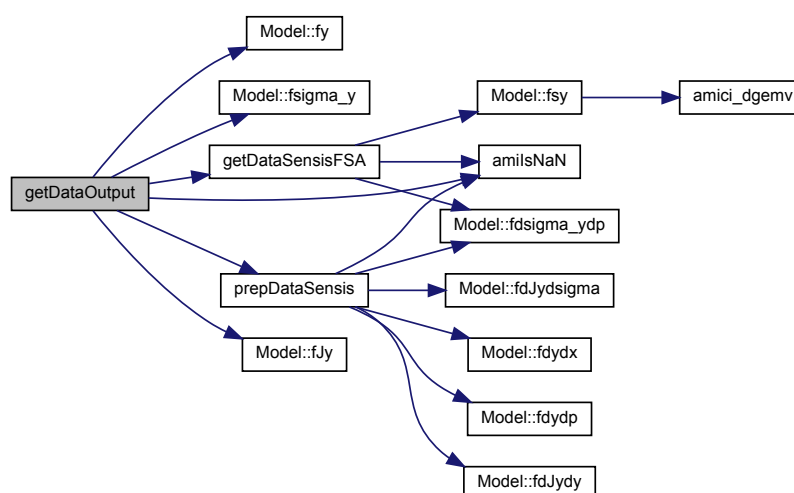
## Returns

status flag indicating success of execution

**Type:** int

Definition at line 690 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.9.2.9 `prepDataSensis()`

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

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

#### Parameters

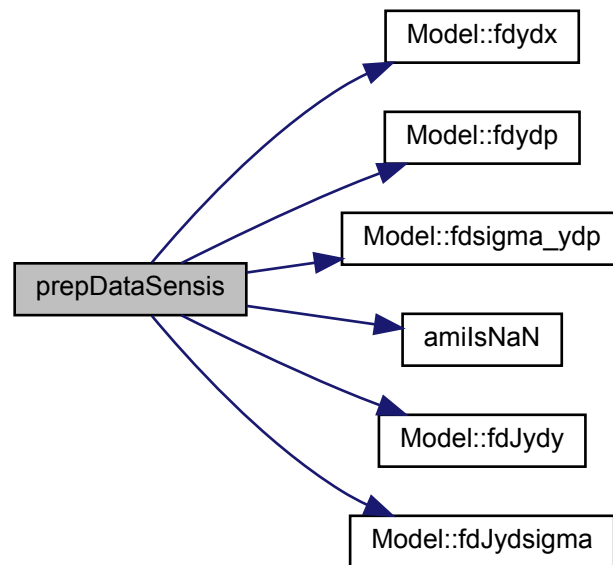
in	<i>it</i>	index of current timepoint <b>Type:</b> int
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

#### Returns

status flag indicating success of execution  
**Type:** int

Definition at line 752 of file `forwardproblem.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.9.2.10 getDataSensisFSA()

```

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

`getDataSensisFSA` extracts data information for forward sensitivity analysis

**Parameters**

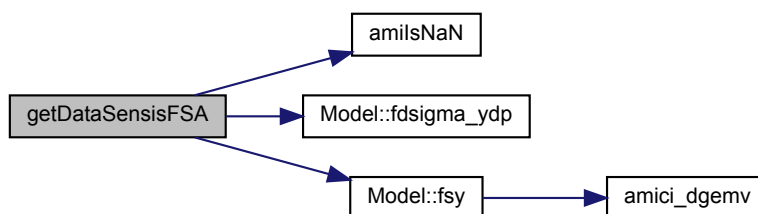
out	<i>status</i>	flag indicating success of execution <b>Type:</b> int
in	<i>it</i>	index of current timepoint <b>Type:</b> int
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>edata</i>	pointer to the experimental data struct <b>Type:</b> <a href="#">ExpData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>solver</i>	pointer to solver object <b>Type:</b> <a href="#">Solver</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

**Returns**

void

Definition at line 833 of file forwardproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.9.2.11 applyEventBolus()

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

applyEventBolus applies the event bolus to the current state

**Parameters**

out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

**Returns**

status flag indicating success of execution  
**Type:** int

Definition at line 906 of file forwardproblem.cpp.

Here is the call graph for this function:

**9.9.2.12 applyEventSensiBolusFSA()**

```

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

applyEventSensiBolusFSA applies the event bolus to the current sensitivities

**Parameters**

out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

**Returns**

status flag indicating success of execution  
**Type:** int

Definition at line 942 of file forwardproblem.cpp.



Here is the call graph for this function:



### 9.9.2.13 updateHeaviside()

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

updateHeaviside updates the heaviside variables h on event occurrences

#### Parameters

--	--

Definition at line 980 of file forwardproblem.cpp.

## 9.10 Model Class Reference

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

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

#### Public Member Functions

- [Model](#) ()
- [Model](#) (const int np, const int nx, const int nxtrue, const int nk, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const AMICI\_o2mode o2mode)
- virtual [Solver](#) \* [getSolver](#) ()
- virtual int [fx0](#) (N\_Vector x0, void \*user\_data)
- virtual int [fdx0](#) (N\_Vector x0, N\_Vector dx0, void \*user\_data)
- virtual int [fsx0](#) (N\_Vector \*sx0, N\_Vector x, N\_Vector dx, void \*user\_data)
- virtual int [fsdx0](#) (N\_Vector \*sdx0, N\_Vector x, N\_Vector dx, void \*user\_data)
- virtual int [fJ](#) (long int N, realtype t, realtype cj, N\_Vector x, N\_Vector dx, N\_Vector xdot, DlsMat J, void \*user\_data, N\_Vector tmp1, N\_Vector tmp2, N\_Vector tmp3)
- virtual int [fJB](#) (long int NeqBdot, realtype t, N\_Vector x, N\_Vector xB, N\_Vector xBdot, DlsMat JB, void \*user\_data, N\_Vector tmp1B, N\_Vector tmp2B, N\_Vector tmp3B)
- virtual int [JJDiag](#) (realtype t, N\_Vector JDiag, N\_Vector x, void \*user\_data)

- virtual int **fJv** (N\_Vector v, N\_Vector Jv, realtype t, N\_Vector x, N\_Vector xdot, void \*user\_data, N\_Vector tmp)
- virtual int **froot** (realtype t, N\_Vector x, N\_Vector dx, realtype \*root, void \*user\_data)
- virtual int **frz** (realtype t, int ie, N\_Vector x, TempData \*tdata, ReturnData \*rdata)
- virtual int **fsrz** (realtype t, int ie, N\_Vector x, N\_Vector \*sx, TempData \*tdata, ReturnData \*rdata)
- virtual int **fstau** (realtype t, int ie, N\_Vector x, N\_Vector \*sx, TempData \*tdata)
- virtual int **fy** (realtype t, int it, N\_Vector x, void \*user\_data, ReturnData \*rdata)
- virtual int **fdydp** (realtype t, int it, N\_Vector x, TempData \*tdata)
- virtual int **fdydx** (realtype t, int it, N\_Vector x, TempData \*tdata)
- virtual int **fz** (realtype t, int ie, N\_Vector x, TempData \*tdata, ReturnData \*rdata)
- virtual int **fsz** (realtype t, int ie, N\_Vector x, N\_Vector \*sx, TempData \*tdata, ReturnData \*rdata)
- virtual int **fdzdp** (realtype t, int ie, N\_Vector x, TempData \*tdata)
- virtual int **fdzdx** (realtype t, int ie, N\_Vector x, TempData \*tdata)
- virtual int **fdrzdp** (realtype t, int ie, N\_Vector x, TempData \*tdata)
- virtual int **fdrzdx** (realtype t, int ie, N\_Vector x, TempData \*tdata)
- virtual int **fxdot** (realtype t, N\_Vector x, N\_Vector dx, N\_Vector xdot, void \*user\_data)
- virtual int **fxBdot** (realtype t, N\_Vector x, N\_Vector dx, N\_Vector xB, N\_Vector dxB, N\_Vector xBdot, void \*user\_data)
- virtual int **fqBdot** (realtype t, N\_Vector x, N\_Vector xB, N\_Vector qBdot, void \*user\_data)
- virtual int **fdxdotdp** (realtype t, N\_Vector x, N\_Vector dx, void \*user\_data)
- virtual int **fdeltax** (realtype t, int ie, N\_Vector x, N\_Vector xdot, N\_Vector xdot\_old, TempData \*tdata)
- virtual int **fdeltasx** (realtype t, int ie, N\_Vector x, N\_Vector xdot, N\_Vector xdot\_old, N\_Vector \*sx, TempData \*tdata)
- virtual int **fdeltaxB** (realtype t, int ie, N\_Vector x, N\_Vector xB, N\_Vector xdot, N\_Vector xdot\_old, TempData \*tdata)
- virtual int **fdeltaqB** (realtype t, int ie, N\_Vector x, N\_Vector xB, N\_Vector qBdot, N\_Vector xdot, N\_Vector xdot\_old, TempData \*tdata)
- virtual int **fsigma\_y** (realtype t, TempData \*tdata)
- virtual int **fdsigma\_ydp** (realtype t, TempData \*tdata)
- virtual int **fsigma\_z** (realtype t, int ie, TempData \*tdata)
- virtual int **fdsigma\_zdp** (realtype t, int ie, TempData \*tdata)
- virtual int **fJy** (realtype t, int it, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fJz** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fJrz** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJydy** (realtype t, int it, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJydsigma** (realtype t, int it, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJzdz** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJzdsigma** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJrzd** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fdJrzdsigma** (realtype t, int ie, N\_Vector x, TempData \*tdata, const ExpData \*edata, ReturnData \*rdata)
- virtual int **fsxdot** (int Ns, realtype t, N\_Vector x, N\_Vector xdot, int ip, N\_Vector sx, N\_Vector sxdot, void \*user\_data, N\_Vector tmp1, N\_Vector tmp2)
- virtual int **fJSparse** (realtype t, N\_Vector x, N\_Vector xdot, SlsMat J, void \*user\_data, N\_Vector tmp1, N\_Vector tmp2, N\_Vector tmp3)
- virtual int **fJBand** (long int N, long int mupper, long int mlower, realtype t, N\_Vector x, N\_Vector xdot, DlsMat J, void \*user\_data, N\_Vector tmp1, N\_Vector tmp2, N\_Vector tmp3)
- virtual int **fJBandB** (long int NeqBdot, long int mupper, long int mlower, realtype t, N\_Vector x, N\_Vector xB, N\_Vector xBdot, DlsMat JB, void \*user\_data, N\_Vector tmp1B, N\_Vector tmp2B, N\_Vector tmp3B)
- virtual int **fJvB** (N\_Vector vB, N\_Vector JvB, realtype t, N\_Vector x, N\_Vector xB, N\_Vector xBdot, void \*user\_data, N\_Vector tmpB)
- virtual int **fJSparseB** (realtype t, N\_Vector x, N\_Vector xB, N\_Vector xBdot, SlsMat JB, void \*user\_data, N\_Vector tmp1B, N\_Vector tmp2B, N\_Vector tmp3B)
- int **fsy** (const int it, const TempData \*tdata, ReturnData \*rdata)

- int **fsz\_tf** (const int ie, const TempData \*tdata, ReturnData \*rdata)
- int **fsJy** (const int it, const TempData \*tdata, ReturnData \*rdata)
- int **fdJydp** (const int it, TempData \*tdata, const ExpData \*edata, const ReturnData \*rdata)
- int **fdJydx** (const int it, TempData \*tdata, const ExpData \*edata)
- int **fsJz** (const int ie, TempData \*tdata, const ReturnData \*rdata)
- int **fdJzdp** (const int ie, TempData \*tdata, const ExpData \*edata, const ReturnData \*rdata)
- int **fdJzdx** (const int ie, TempData \*tdata, const ExpData \*edata)
- int **initialize** (const UserData \*udata, TempData \*tdata)
- int **initializeStates** (const double \*x0data, TempData \*tdata)
- int **initHeaviside** (TempData \*tdata)

#### Public Attributes

- const int **np**
- const int **nk**
- const int **nx**
- const int **nxtrue**
- const int **ny**
- const int **nytrue**
- const int **nz**
- const int **nztrue**
- const int **ne**
- const int **nw**
- const int **ndwdx**
- const int **ndwdp**
- const int **nnz**
- const int **nJ**
- const int **ubw**
- const int **lbw**
- const AMICI\_o2mode **o2mode**
- int \* **z2event** = nullptr
- realtype \* **idlist** = nullptr

#### 9.10.1 Detailed Description

Definition at line 16 of file amici\_model.h.

#### 9.10.2 Constructor & Destructor Documentation

##### 9.10.2.1 Model() [1/2]

Model ( )

default constructor

Definition at line 19 of file amici\_model.h.

### 9.10.2.2 Model() [2/2]

```

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

```

constructor with model dimensions

#### Parameters

<i>np</i>	number of parameters
<i>nx</i>	number of state variables
<i>nxtrue</i>	number of state variables of the non-augmented model
<i>nk</i>	number of constants
<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

Definition at line 43 of file amici\_model.h.

### 9.10.3 Member Function Documentation

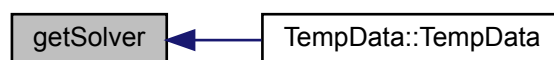
9.10.3.1 `getSolver()`

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

Retrieves the solver object

Definition at line 51 of file amici\_model.h.

Here is the caller graph for this function:

9.10.3.2 `fx0()`

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

Initial states

## Parameters

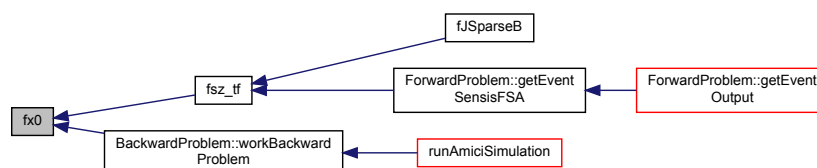
out	<i>x0</i>	Vector to which the initial states will be written <b>Type:</b> <code>N_Vector</code>
in	<i>user_data</i>	object with model specifications <b>Type:</b> <code>TempData</code>

## Returns

status flag indicating successful execution  
**Type:** `int`

Definition at line 58 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.3 fdx0()

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

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

#### Parameters

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

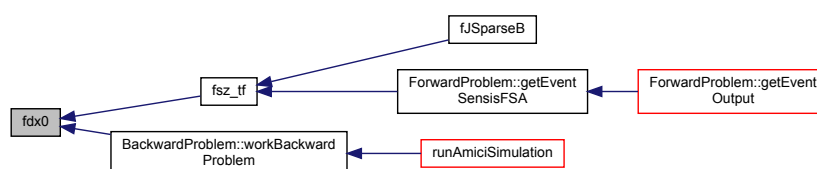
#### Returns

status flag indicating successful execution

**Type:** int

Definition at line 66 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.4 fsx0()

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

Initial value for initial state sensitivities

## Parameters

out	<i>sx0</i>	Vector to which the initial state sensitivities <b>Type:</b> N_Vector
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>dx</i>	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
in	<i>user_data</i>	object with model specifications <b>Type:</b> TempData

## Returns

status flag indicating successful execution

**Type:** int

Definition at line 77 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.5 fsdx0()

```

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

```

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

## Parameters

out	<i>sdx0</i>	Vector to which the derivative initial state sensitivities <b>Type:</b> N_Vector
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>dx</i>	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
in	<i>user_data</i>	object with model specifications <b>Type:</b> TempData

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 86 of file amici\_model.h.

**9.10.3.6 fJ()**

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

Jacobian of xdot with respect to states x

**Parameters**

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

**Returns**

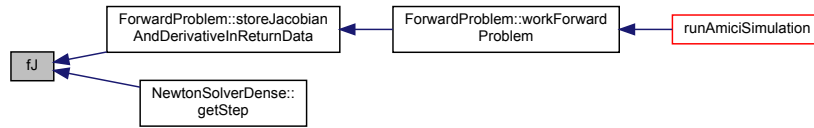
status flag indicating successful execution

**Type:** int



Definition at line 102 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.7 fJB()

```

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

```

Jacobian of `xBdot` with respect to adjoint state `xB`

#### Parameters

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

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 117 of file amici\_model.h.

**9.10.3.8 fJDiag()**

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

diagonalized Jacobian (for preconditioning)

**Parameters**

in	<i>t</i>	timepoint <b>Type:</b> realtype
out	<i>JDiag</i>	Vector to which the Jacobian diagonal will be written <b>Type:</b> N_Vector
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>user_data</i>	object with model specifications <b>Type:</b> TempData

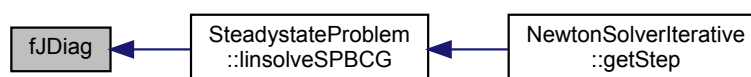
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 126 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.9 fJv()

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

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

## Parameters

in	<i>v</i>	Vector with which the Jacobian is multiplied <b>Type:</b> N_Vector
out	<i>Jv</i>	Vector to which the Jacobian vector product will be written <b>Type:</b> N_Vector
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
in	<i>user_data</i>	object with model specifications <b>Type:</b> TempData
in	<i>tmp</i>	temporary storage vector <b>Type:</b> N_Vector

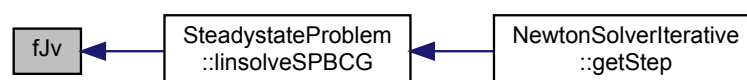
## Returns

status flag indicating successful execution

**Type:** int

Definition at line 138 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.10 froot()

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

Event trigger function for events

#### Parameters

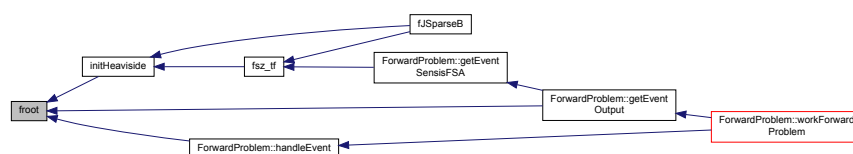
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>dx</i>	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
out	<i>root</i>	array with root function values <b>Type:</b> realtype
in	<i>user_data</i>	object with model specifications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

#### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 148 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.11 frz()

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

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

## Parameters

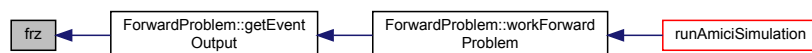
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in, out	<i>rdata</i>	pointer to return data object <b>Type:</b> ReturnData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 158 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.12 ffsrz()

```

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

Sensitivity of rz, total derivative

## Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>sx</i>	Vector with the state sensitivities <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
Generated by Doxygen		
in, out	<i>rdata</i>	pointer to return data object <b>Type:</b> ReturnData

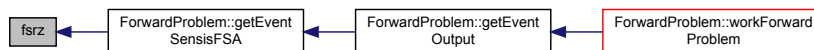
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 169 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.13 ftau()**

```

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

Sensitivity of event timepoint, total derivative

**Parameters**

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

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 179 of file amici\_model.h.

9.10.3.14 `fy()`

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

Observables / measurements

## Parameters

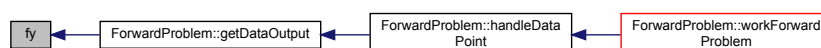
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>it</i>	timepoint index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>user_data</i>	pointer to temp data object <b>Type:</b> TempData
in, out	<i>rdata</i>	pointer to return data object <b>Type:</b> ReturnData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 189 of file amici\_model.h.

Here is the caller graph for this function:

9.10.3.15 `fdydp()`

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

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

**Parameters**

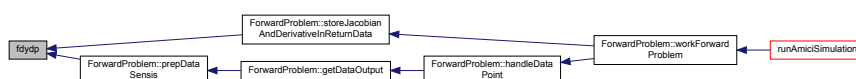
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>it</i>	timepoint index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in, out	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

**Returns**

status flag indicating successful execution  
**Type:** int

Definition at line 198 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.16 fdydx()**

```

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

```

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

**Parameters**

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



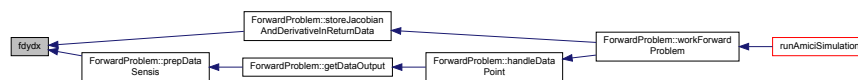
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 207 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.17 fz()**

```

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

Event-resolved measurements

**Parameters**

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

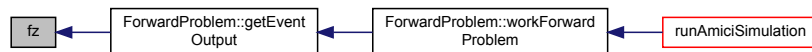
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 217 of file amici\_model.h.

Here is the caller graph for this function:



#### 9.10.3.18 fsz()

```

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

Sensitivity of z, total derivative

##### Parameters

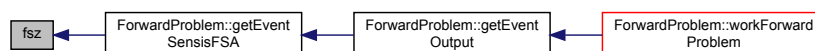
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>sx</i>	Vector with the state sensitiviies <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in, out	<i>rdata</i>	pointer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

##### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 228 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.19 fdzdp()

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

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

## Parameters

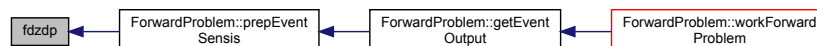
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in, out	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 237 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.20 fdzdx()

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

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

## Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
Generated by Doxygen		Vector with the states <b>Type:</b> N_Vector
in, out	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

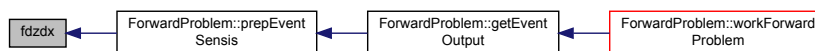
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 246 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.21 fdrzdp()**

```

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

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

**Parameters**

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

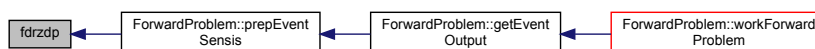
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 255 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.22 fdrzdx()

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

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

## Parameters

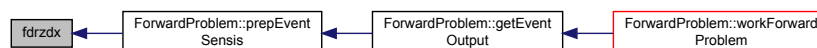
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in, out	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 264 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.23 fxdot()

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

Right hand side of differential equation for states x

## Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
----	----------	------------------------------------

**Parameters**

in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$dx$	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
out	$xdot$	Vector with the right hand side <b>Type:</b> N_Vector
in	$user\_data$	pointer to temp data object <b>Type:</b> TempData

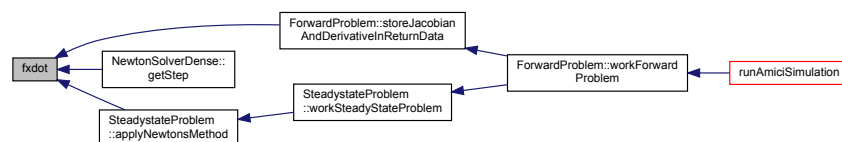
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 274 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.24 fxBdot()**

```

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

```

Right hand side of differential equation for adjoint state xB

**Parameters**

in	$t$	timepoint <b>Type:</b> realtype
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$dx$	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
in	$xB$	Vector with the adjoint states <b>Type:</b> N_Vector

## Parameters

in	$dxB$	Vector with the adjoint derivative states (only DAE) <b>Type:</b> N_Vector
out	$xBdot$	Vector with the adjoint right hand side <b>Type:</b> N_Vector
in	$user\_data$	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution

**Type:** int

Definition at line 286 of file amici\_model.h.

## 9.10.3.25 fqBdot()

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

Right hand side of integral equation for quadrature states qB

## Parameters

in	$t$	timepoint <b>Type:</b> realtype
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$xB$	Vector with the adjoint states <b>Type:</b> N_Vector
out	$qBdot$	Vector with the adjoint quadrature right hand side <b>Type:</b> N_Vector
in	$user\_data$	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution

**Type:** int

Definition at line 296 of file amici\_model.h.

### 9.10.3.26 fdxdotdp()

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

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

#### Parameters

in	$t$	timepoint <b>Type:</b> realtype
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$dx$	Vector with the derivative states (only DAE) <b>Type:</b> N_Vector
in	$user\_data$	pointer to temp data object <b>Type:</b> TempData

#### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 305 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.27 fdeltax()

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

State update functions for events



## Parameters

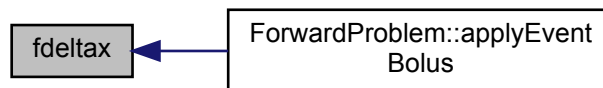
in	$t$	timepoint <b>Type:</b> realtype
in	$ie$	event index <b>Type:</b> int
in, out	$x$	Vector with the states <b>Type:</b> N_Vector
in	$\dot{x}$	Vector with the right hand side <b>Type:</b> N_Vector
in	$\dot{x}_{old}$	Vector with the right hand side before the event <b>Type:</b> N_Vector
in	$tdata$	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 316 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.28 fdeltasx()

```

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

Sensitivity update functions for events, total derivative

**Parameters**

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
in	<i>xdot_old</i>	Vector with the right hand side before the event <b>Type:</b> N_Vector
in	<i>sx</i>	Vector with the state sensitivities <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

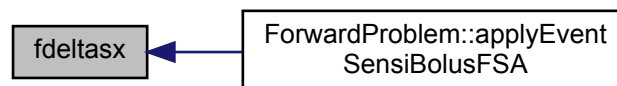
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 328 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.29 fdeltaxB()**

```

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

Adjoint state update functions for events

## Parameters

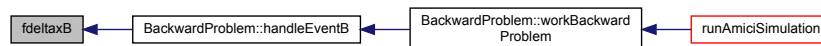
in	$t$	timepoint <b>Type:</b> realtype
in	$ie$	event index <b>Type:</b> int
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$xB$	Vector with the adjoint states <b>Type:</b> N_Vector
in	$xdot$	Vector with the right hand side <b>Type:</b> N_Vector
in	$xdot\_old$	Vector with the right hand side before the event <b>Type:</b> N_Vector
in	$tdata$	pointer to temp data object <b>Type:</b> TempData

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 340 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.30 fdeltaqB()

```

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

Quadrature state update functions for events

## Parameters

in	$t$	timepoint <b>Type:</b> realtype
----	-----	------------------------------------

**Parameters**

in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xB</i>	Vector with the adjoint states <b>Type:</b> N_Vector
in	<i>qBdot</i>	Vector with the adjoint quadrature states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
in	<i>xdot_old</i>	Vector with the right hand side before the event <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

**Returns**

status flag indicating successful execution  
**Type:** int

Definition at line 353 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.31 fsigma\_y()**

```

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

Standard deviation of measurements

**Parameters**

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 360 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.32 fdsigma\_ydp()**

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

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

**Parameters**

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

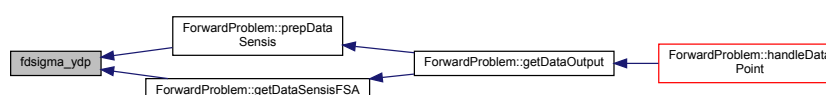
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 367 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.33 fsigma\_z()

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

Standard deviation of events

#### Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

#### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 375 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.34 fdsigma\_zdp()

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

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

#### Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 383 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.35 fJy()**

```

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

negative log-likelihood of time-resolved measurements y

**Parameters**

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

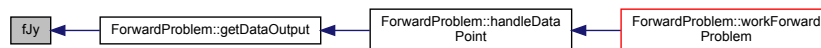
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 394 of file amici\_model.h.

Here is the caller graph for this function:



### 9.10.3.36 fJz()

```

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

negative log-likelihood of event-resolved measurements z

#### Parameters

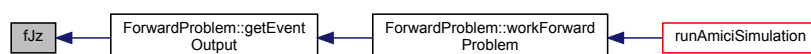
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> ExpData
in, out	<i>rdata</i>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

#### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 405 of file amici\_model.h.

Here is the caller graph for this function:





## 9.10.3.37 fJrz()

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

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

## Parameters

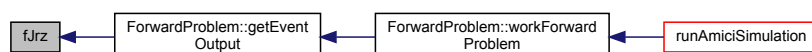
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> ExpData
in, out	<i>rdata</i>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

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 416 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.38 fdJydy()

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

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

## Parameters

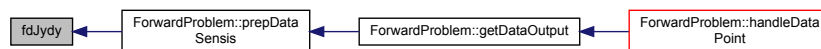
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>it</i>	timepoint index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> ExpData
in, out	<i>rdata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

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 427 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.39 fdJydsigma()

```

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

```

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

## Parameters

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

**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 438 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.40 fdJzdz()**

```

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

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

**Parameters**

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

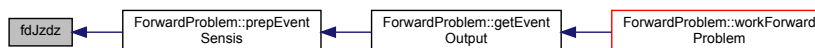
**Returns**

status flag indicating successful execution

**Type:** int

Definition at line 449 of file amici\_model.h.

Here is the caller graph for this function:



#### 9.10.3.41 fdJzdsigma()

```

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

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

##### Parameters

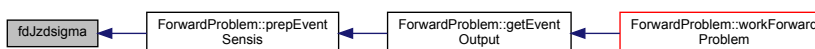
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> ExpData
in, out	<i>rdata</i>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

##### Returns

status flag indicating successful execution  
**Type:** int

Definition at line 460 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.42 fdJrzdz()

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

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

## Parameters

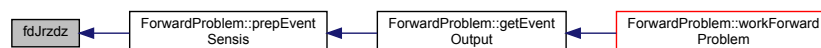
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>ie</i>	event index <b>Type:</b> int
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> ExpData
in, out	<i>rdata</i>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

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 471 of file amici\_model.h.

Here is the caller graph for this function:



## 9.10.3.43 fdJrzdsigma()

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

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

**Parameters**

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

**Returns**

status flag indicating successful execution  
**Type:** int

Definition at line 482 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.44 fsxdot()**

```

virtual int fsxdot (
    int Ns,
    realtype t,
    N_Vector x,
    N_Vector xdot,
    int ip,
    N_Vector sx,
    N_Vector sxdot,
    void * user_data,
    N_Vector tmp1,
    N_Vector tmp2 ) [virtual]
  
```

Right hand side of differential equation for state sensitivities *sx*

**Parameters**

in	<i>Ns</i>	number of parameters <b>Type:</b> int
----	-----------	--

## Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
in	<i>ip</i>	parameter index <b>Type:</b> int
in	<i>sx</i>	Vector with the state sensitivities <b>Type:</b> N_Vector
in	<i>sxdot</i>	Vector with the sensitivity right hand side <b>Type:</b> N_Vector
in	<i>user_data</i>	pointer to temp data object <b>Type:</b> TempData
in	<i>tmp1</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp2</i>	temporary storage vector <b>Type:</b> N_Vector

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 497 of file amici\_model.h.

## 9.10.3.45 fJSparse()

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

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

## Parameters

in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
out	<i>J</i>	Matrix to which the Jacobian will be written <b>Type:</b> SlsMat

**Parameters**

in	<i>user_data</i>	object with model specifications <b>Type:</b> <a href="#">TempData</a>
in	<i>tmp1</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp2</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp3</i>	temporary storage vector <b>Type:</b> N_Vector

**Returns**

status flag indicating successful execution  
**Type:** int

Definition at line 510 of file amici\_model.h.

Here is the caller graph for this function:

**9.10.3.46 fJBand()**

```

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

J in banded form (for banded solvers)

**Parameters**

in	<i>N</i>	number of states <b>Type:</b> long int
----	----------	---



## Parameters

in	<i>mupper</i>	upper matrix bandwidth <b>Type:</b> long int
in	<i>mlower</i>	lower matrix bandwidth <b>Type:</b> long int
in	<i>t</i>	timepoint <b>Type:</b> realtype
in	<i>x</i>	Vector with the states <b>Type:</b> N_Vector
in	<i>xdot</i>	Vector with the right hand side <b>Type:</b> N_Vector
out	<i>J</i>	Matrix to which the Jacobian will be written <b>Type:</b> DlsMat
in	<i>user_data</i>	object with model specifications <b>Type:</b> <a href="#">TempData</a>
in	<i>tmp1</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp2</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp3</i>	temporary storage vector <b>Type:</b> N_Vector

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 526 of file amici\_model.h.

## 9.10.3.47 fJBandB()

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

JB in banded form (for banded solvers)

## Parameters

in	<i>NeqBdot</i>	number of states <b>Type:</b> long int
----	----------------	---

## Parameters

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

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 543 of file amici\_model.h.

## 9.10.3.48 fJvB()

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

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

## Parameters

in	<i>vB</i>	Vector with which the Jacobian is multiplied <b>Type:</b> N_Vector
out	<i>JvB</i>	Vector to which the Jacobian vector product will be written <b>Type:</b> N_Vector

## Parameters

in	$t$	timepoint <b>Type:</b> realtype
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$xB$	Vector with the adjoint states <b>Type:</b> N_Vector
in	$xBdot$	Vector with the adjoint right hand side <b>Type:</b> N_Vector
in	$user\_data$	object with model specifications <b>Type:</b> TempData
in	$tmpB$	temporary storage vector <b>Type:</b> N_Vector

## Returns

status flag indicating successful execution  
**Type:** int

Definition at line 556 of file amici\_model.h.

## 9.10.3.49 fJSparseB()

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

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

## Parameters

in	$t$	timepoint <b>Type:</b> realtype
in	$x$	Vector with the states <b>Type:</b> N_Vector
in	$xB$	Vector with the adjoint states <b>Type:</b> N_Vector
in	$xBdot$	Vector with the adjoint right hand side <b>Type:</b> N_Vector
out	$JB$	Matrix to which the Jacobian will be written <b>Type:</b> DlsMat
in	$user\_data$	object with model specifications <b>Type:</b> TempData

## Parameters

in	<i>tmp1B</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp2B</i>	temporary storage vector <b>Type:</b> N_Vector
in	<i>tmp3B</i>	temporary storage vector <b>Type:</b> N_Vector

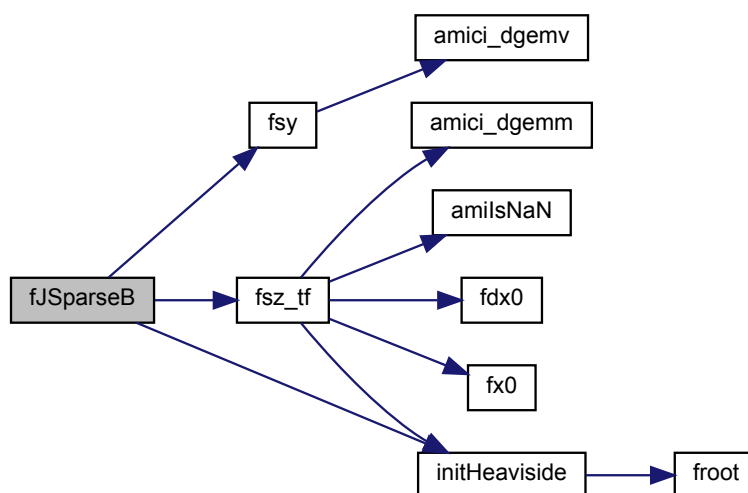
## Returns

status flag indicating successful execution

**Type:** int

Definition at line 570 of file amici\_model.h.

Here is the call graph for this function:



## 9.10.3.50 fsy()

```

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

```

Sensitivity of measurements y, total derivative

## Parameters

in	<i>it</i>	timepoint index <b>Type:</b> int
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> <a href="#">TempData</a>
in, out	<i>rdata</i>	pointer to return data object <b>Type:</b> <a href="#">ReturnData</a>

## Returns

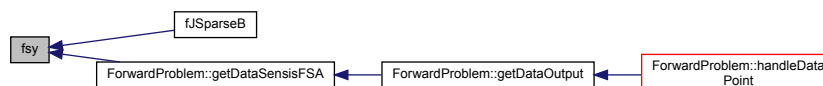
status flag indicating successful execution  
**Type:** int

Definition at line 30 of file amici\_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.10.3.51 fsz\_tf()

```

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

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

## Parameters

in	<i>ie</i>	event index <b>Type:</b> int
in	<i>tdata</i>	pointer to temp data object <b>Type:</b> <a href="#">TempData</a>
<b>Generated by Doxygen</b>		
in, out	<i>rdata</i>	pointer to return data object <b>Type:</b> <a href="#">ReturnData</a>

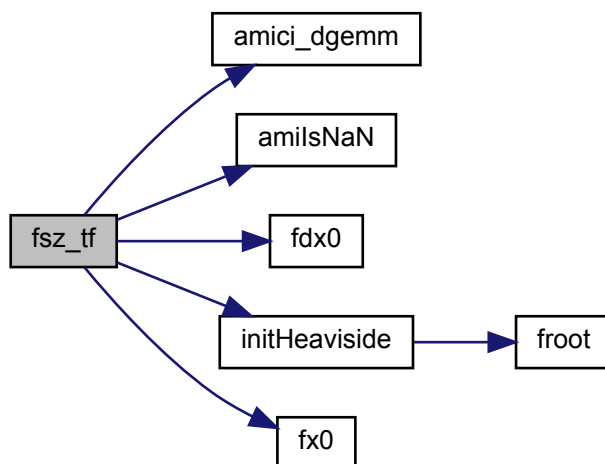
**Returns**

status flag indicating successful execution

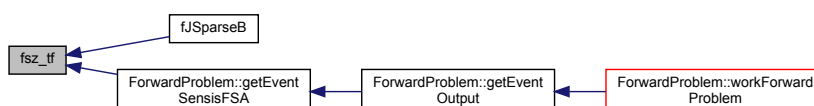
**Type:** int

Definition at line 58 of file amici\_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.10.3.52 initHeaviside()

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

`initHeaviside` initialises the heaviside variables `h` at the initial time `t0` heaviside variables activate/deactivate on event occurrences

## Parameters

out	tdata	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
-----	-------	---

## Returns

status flag indicating success of execution

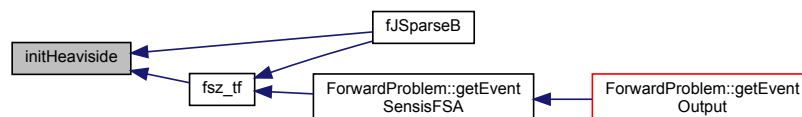
**Type:** int

Definition at line 470 of file amici\_model.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.10.4 Member Data Documentation

## 9.10.4.1 np

```
const int np
```

total number of model parameters

Definition at line 599 of file amici\_model.h.

**9.10.4.2 nk**

```
const int nk
```

number of fixed parameters

Definition at line 601 of file amici\_model.h.

**9.10.4.3 nx**

```
const int nx
```

number of states

Definition at line 603 of file amici\_model.h.

**9.10.4.4 nxtrue**

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 605 of file amici\_model.h.

**9.10.4.5 ny**

```
const int ny
```

number of observables

Definition at line 607 of file amici\_model.h.

**9.10.4.6 nytrue**

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 609 of file amici\_model.h.



**9.10.4.7 nz**

```
const int nz
```

number of event outputs

Definition at line 611 of file amici\_model.h.

**9.10.4.8 nztrue**

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 613 of file amici\_model.h.

**9.10.4.9 ne**

```
const int ne
```

number of events

Definition at line 615 of file amici\_model.h.

**9.10.4.10 nw**

```
const int nw
```

number of common expressions

Definition at line 617 of file amici\_model.h.

**9.10.4.11 ndwdx**

```
const int ndwdx
```

number of derivatives of common expressions wrt x

Definition at line 619 of file amici\_model.h.

**9.10.4.12 ndwdp**

```
const int ndwdp
```

number of derivatives of common expressions wrt p

Definition at line 621 of file amici\_model.h.

**9.10.4.13 nnz**

```
const int nnz
```

number of nonzero entries in jacobian

Definition at line 623 of file amici\_model.h.

**9.10.4.14 nJ**

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 625 of file amici\_model.h.

**9.10.4.15 ubw**

```
const int ubw
```

upper bandwidth of the jacobian

Definition at line 627 of file amici\_model.h.

**9.10.4.16 lbw**

```
const int lbw
```

lower bandwidth of the jacobian

Definition at line 629 of file amici\_model.h.

#### 9.10.4.17 o2mode

```
const AMICI_o2mode o2mode
```

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

Definition at line 632 of file `amici_model.h`.

#### 9.10.4.18 z2event

```
int* z2event = nullptr
```

index indicating to which event an event output belongs

Definition at line 634 of file `amici_model.h`.

#### 9.10.4.19 idlist

```
realtype* idlist = nullptr
```

flag array for DAE equations

Definition at line 636 of file `amici_model.h`.

## 9.11 modelTest Class Reference

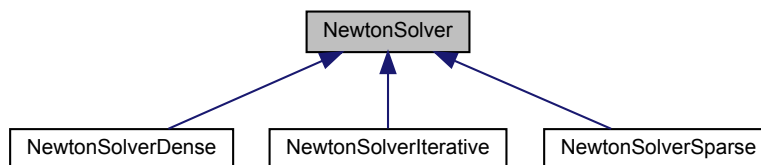
MODELTEST Summary of this class goes here Detailed explanation goes here.

### 9.11.1 Detailed Description

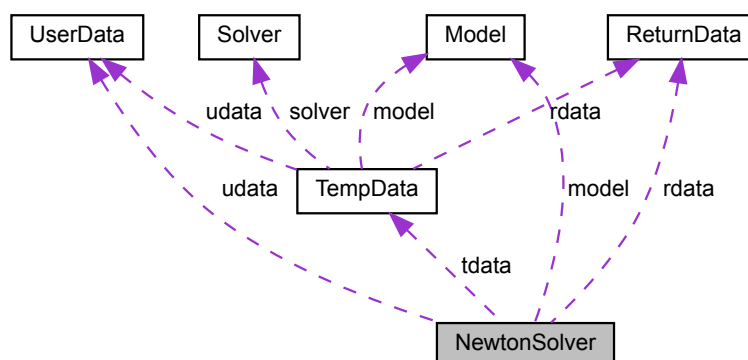
Definition at line 17 of file `modelTest.m`.

## 9.12 NewtonSolver Class Reference

Inheritance diagram for NewtonSolver:



Collaboration diagram for NewtonSolver:



### Public Member Functions

- **NewtonSolver** (`Model` \*model, `ReturnData` \*rdata, `UserData` \*udata, `TempData` \*tdata)
- virtual int `getStep` (int ntry, int nnewt, N\_Vector delta)=0

### Static Public Member Functions

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

### Protected Attributes

- `Model` \* **model**
- `ReturnData` \* **rdata**
- `UserData` \* **udata**
- `TempData` \* **tdata**

## 9.12.1 Detailed Description

Definition at line 17 of file newton\_solver.h.

## 9.12.2 Member Function Documentation

## 9.12.2.1 getSolver()

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

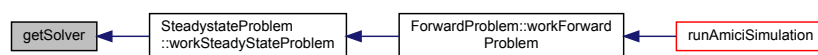
getNewtonStep computes the Newton Step by solving the linear system

## Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>status</i>	flag indicating success of execution <b>Type:</b> int

Definition at line 18 of file newton\_solver.cpp.

Here is the caller graph for this function:



## 9.12.2.2 getStep()

```
virtual int getStep (
    int ntry,
    int nnewt,
    N_Vector delta ) [pure virtual]
```

**Parameters**

in	<i>ntry</i>	integer number of Newton solver try
in	<i>nnewt</i>	integer number of Newton steps in the current Newton solver try
out	<i>delta</i>	N_Vector solution of the linear system

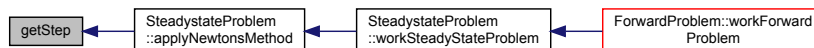
**Returns**

int status flag indicating success of execution

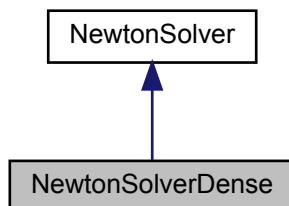
**Type:** int

Implemented in [NewtonSolverIterative](#), [NewtonSolverSparse](#), and [NewtonSolverDense](#).

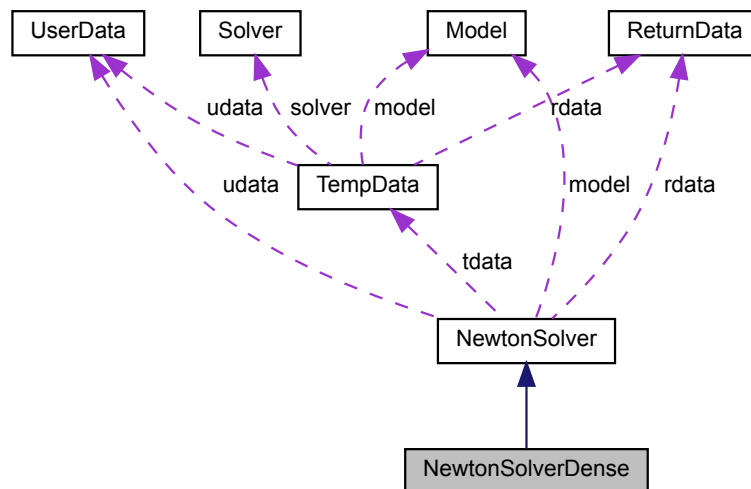
Here is the caller graph for this function:

**9.13 NewtonSolverDense Class Reference**

Inheritance diagram for NewtonSolverDense:



Collaboration diagram for NewtonSolverDense:



#### Public Member Functions

- **NewtonSolverDense** ([Model](#) \*model, [ReturnData](#) \*rdata, [UserData](#) \*udata, [TempData](#) \*tdata)
- int [getStep](#) (int ntry, int nnewt, N\_Vector delta)

#### Additional Inherited Members

##### 9.13.1 Detailed Description

Definition at line 42 of file `newton_solver.h`.

##### 9.13.2 Member Function Documentation

###### 9.13.2.1 getStep()

```

int getStep (
    int ntry,
    int nnewt,
    N_Vector delta ) [virtual]

```

#### Parameters

in	<i>ntry</i>	integer number of Newton solver try
in	<i>nnewt</i>	integer number of Newton steps in the current Newton solver try
out	<i>delta</i>	N_Vector solution of the linear system

**Returns**

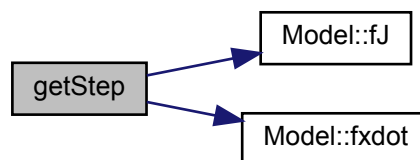
int status flag indicating success of execution

**Type:** int

Implements [NewtonSolver](#).

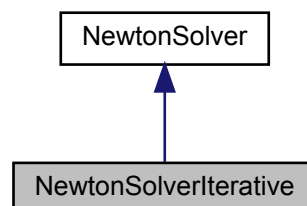
Definition at line 95 of file newton\_solver.cpp.

Here is the call graph for this function:



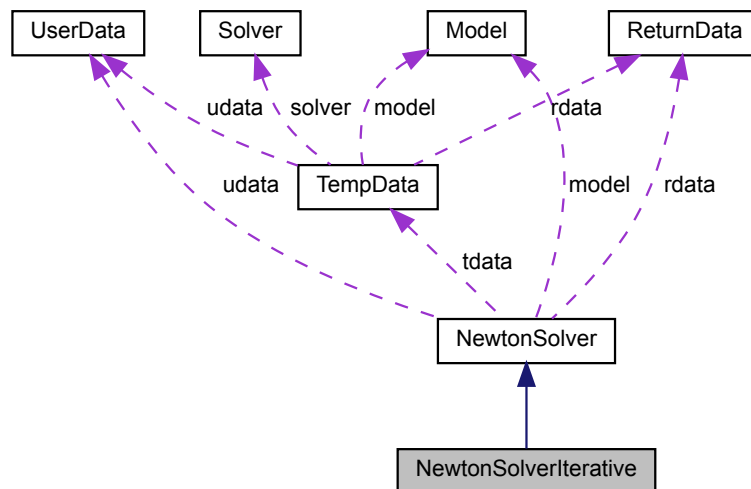
## 9.14 NewtonSolverIterative Class Reference

Inheritance diagram for NewtonSolverIterative:





Collaboration diagram for NewtonSolverIterative:



### Public Member Functions

- **NewtonSolverIterative** ([Model](#) \*model, [ReturnData](#) \*rdata, [UserData](#) \*udata, [TempData](#) \*tdata)
- int [getStep](#) (int ntry, int nnewt, N\_Vector delta)

### Additional Inherited Members

#### 9.14.1 Detailed Description

Definition at line 77 of file newton\_solver.h.

#### 9.14.2 Member Function Documentation

##### 9.14.2.1 getStep()

```

int getStep (
    int ntry,
    int nnewt,
    N_Vector delta ) [virtual]

```

#### Parameters

in	<i>ntry</i>	integer number of Newton solver try
in	<i>nnewt</i>	integer number of Newton steps in the current Newton solver try
out	<i>delta</i>	N_Vector solution of the linear system

**Returns**

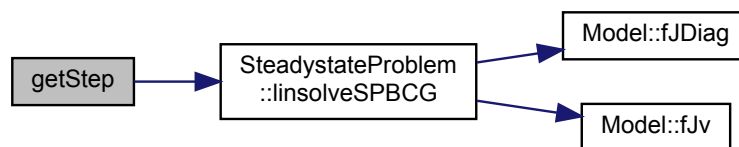
int status flag indicating success of execution

**Type:** int

Implements [NewtonSolver](#).

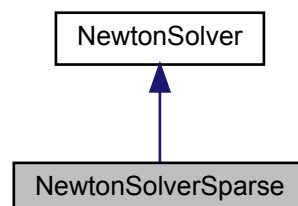
Definition at line 189 of file newton\_solver.cpp.

Here is the call graph for this function:

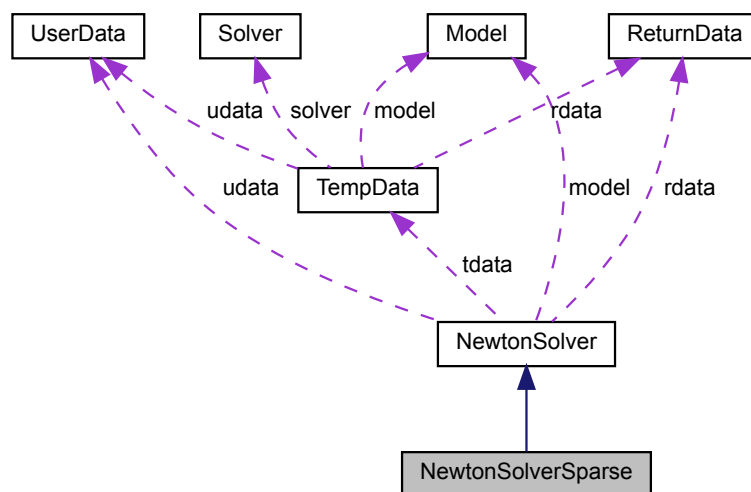


## 9.15 NewtonSolverSparse Class Reference

Inheritance diagram for `NewtonSolverSparse`:



Collaboration diagram for NewtonSolverSparse:



#### Public Member Functions

- **NewtonSolverSparse** ([Model](#) \*model, [ReturnData](#) \*rdata, [UserData](#) \*udata, [TempData](#) \*tdata)
- int [getStep](#) (int ntry, int nnewt, N\_Vector delta)

#### Additional Inherited Members

##### 9.15.1 Detailed Description

Definition at line 58 of file newton\_solver.h.

##### 9.15.2 Member Function Documentation

###### 9.15.2.1 getStep()

```

int getStep (
    int ntry,
    int nnewt,
    N_Vector delta ) [virtual]

```

#### Parameters

in	<i>ntry</i>	integer number of Newton solver try
in	<i>nnewt</i>	integer number of Newton steps in the current Newton solver try
out	<i>delta</i>	N_Vector solution of the linear system

**Returns**

int status flag indicating success of execution

**Type:** int

Implements [NewtonSolver](#).

Definition at line 142 of file newton\_solver.cpp.

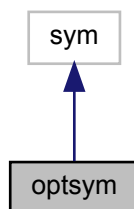
Here is the call graph for this function:



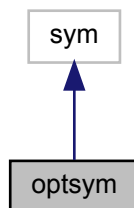
## 9.16 optsym Class Reference

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`:



Collaboration 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*

## 9.16.1 Detailed Description

Definition at line 17 of file optsym.m.

## 9.16.2 Constructor &amp; Destructor Documentation

## 9.16.2.1 optsym()

```
optsym (
    ::sym symbol )
```

## Parameters

<i>symbol</i>	symbolic object
---------------	-----------------

Definition at line 32 of file optsym.m.

## 9.16.3 Member Function Documentation

## 9.16.3.1 getoptimized()

```
mlhsInnerSubst<::sym > getoptimized ( )
```

## Return values

<i>out</i>	optimized symbolic object
------------	---------------------------

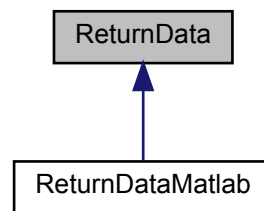
Definition at line 42 of file optsym.m.

## 9.17 ReturnData Class Reference

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

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

Inheritance diagram for ReturnData:



#### Public Member Functions

- [ReturnData](#) ()  
*default constructor*
- [ReturnData](#) (const [UserData](#) \*udata, const [Model](#) \*model)
- virtual void [setDefaults](#) ()
- void [invalidate](#) ()
- void [setLikelihoodSensitivityFirstOrderNaN](#) ()
- void [setLikelihoodSensitivitySecondOrderNaN](#) ()
- int [applyChainRuleFactorToSimulationResults](#) (const [UserData](#) \*udata, const realtype \*unscaledParameters)
- virtual [~ReturnData](#) ()

#### Public Attributes

- double \* [ts](#)
- double \* [xdot](#)
- double \* [dxdotdp](#)
- double \* [dydx](#)
- double \* [dydp](#)
- double \* [J](#)
- double \* [z](#)
- double \* [sigmaz](#)
- double \* [sz](#)
- double \* [ssigmaz](#)
- double \* [rz](#)
- double \* [srz](#)
- double \* [s2rz](#)
- double \* [x](#)
- double \* [sx](#)
- double \* [y](#)
- double \* [sigmay](#)
- double \* [sy](#)
- double \* [ssigmay](#)
- double \* [numsteps](#)
- double \* [numstepsB](#)
- double \* [numrhsevals](#)
- double \* [numrhsevalsB](#)

- double \* [numerrtestfails](#)
- double \* [numerrtestfailsB](#)
- double \* [numnonlinsolvconvfails](#)
- double \* [numnonlinsolvconvfailsB](#)
- double \* [order](#)
- double \* [newton\\_status](#)
- double \* [newton\\_time](#)
- double \* [newton\\_numsteps](#)
- double \* [newton\\_numlinsteps](#)
- double \* [xss](#)
- double \* [llh](#)
- double \* [chi2](#)
- double \* [sllh](#)
- double \* [s2llh](#)
- double \* [status](#)
- const int [np](#)
- const int [nk](#)
- const int [nx](#)
- const int [nxtrue](#)
- const int [ny](#)
- const int [nytrue](#)
- const int [nz](#)
- const int [nztrue](#)
- const int [ne](#)
- const int [nJ](#)
- const int [nplist](#)
- const int [nmaxevent](#)
- const int [nt](#)
- const int [newton\\_maxsteps](#)
- const AMICI\_parameter\_scaling [pscale](#)
- const AMICI\_o2mode [o2mode](#)
- const AMICI\_sensi\_order [sensi](#)
- const AMICI\_sensi\_meth [sensi\\_meth](#)

#### Protected Member Functions

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

#### Protected Attributes

- bool [freeFieldsOnDestruction](#)

#### 9.17.1 Detailed Description

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

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

## 9.17.2 Constructor & Destructor Documentation

### 9.17.2.1 ReturnData()

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

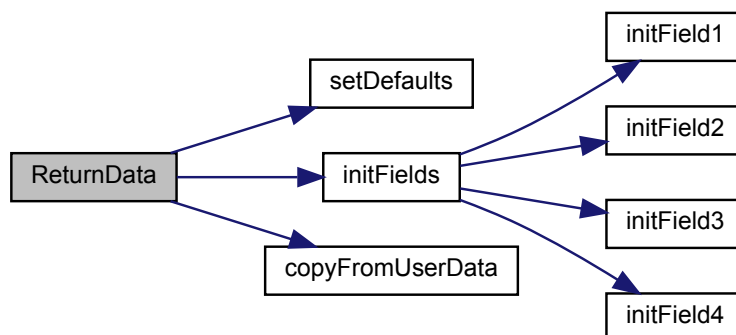
constructor that uses information from model and userdata to appropriately initialize fields

#### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

Definition at line 19 of file rdata.cpp.

Here is the call graph for this function:



### 9.17.2.2 ~ReturnData()

```
~ReturnData ( ) [virtual]
```

default destructor

Definition at line 268 of file rdata.cpp.



## 9.17.3 Member Function Documentation

## 9.17.3.1 setDefaults()

```
void setDefaults ( ) [virtual]
```

initialize all member fields with nullpointers

Definition at line 40 of file rdata.cpp.

Here is the caller graph for this function:



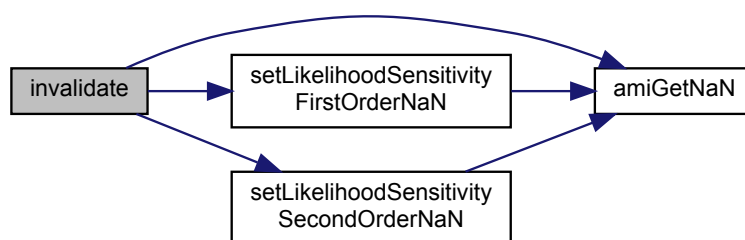
## 9.17.3.2 invalidate()

```
void invalidate ( )
```

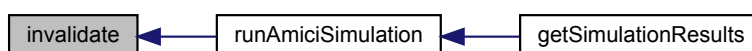
routine to set likelihood and respective sensitivities to NaN (typically after integration failure)

Definition at line 57 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.3 `setLikelihoodSensitivityFirstOrderNaN()`

```
void setLikelihoodSensitivityFirstOrderNaN ( )
```

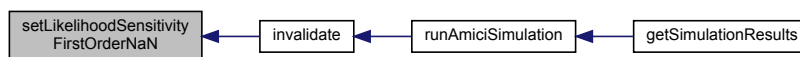
routine to set first order sensitivities to NaN (typically after integration failure)

Definition at line 71 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.4 `setLikelihoodSensitivitySecondOrderNaN()`

```
void setLikelihoodSensitivitySecondOrderNaN ( )
```

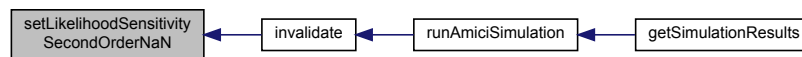
routine to set second order sensitivities to NaN (typically after integration failure)

Definition at line 78 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.5 applyChainRuleFactorToSimulationResults()

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

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

#### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <code>UserData</code>
in	<i>unscaledParameters</i>	pointer to the non-transformed parameters <b>Type:</b> <code>realtype</code>

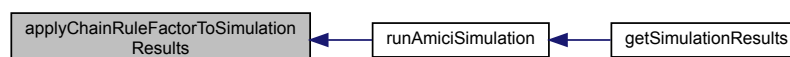
#### Returns

status flag indicating success of execution

**Type:** `int`

Definition at line 85 of file `rdata.cpp`.

Here is the caller graph for this function:



### 9.17.3.6 copyFromUserData()

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

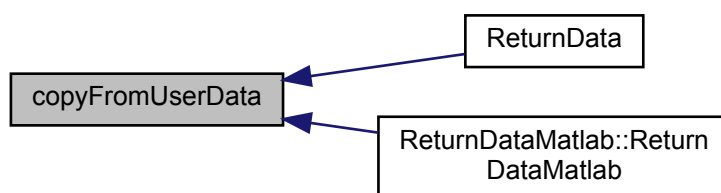
copies measurement timepoints from `UserData` object

**Parameters**

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
----	--------------	--

Definition at line 353 of file rdata.cpp.

Here is the caller graph for this function:

**9.17.3.7 initFields()**

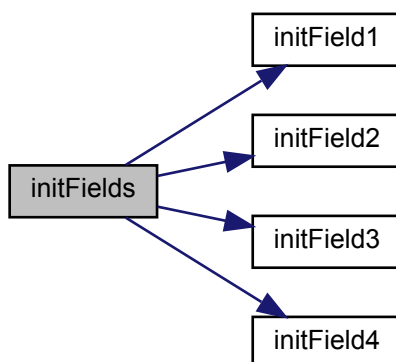
```
void initFields ( ) [protected], [virtual]
```

initialises sol object with the corresponding fields

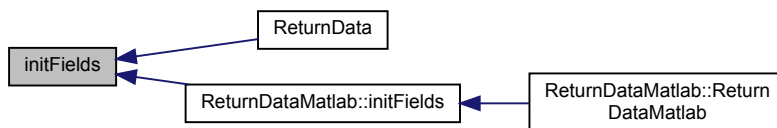
Reimplemented in [ReturnDataMatlab](#).

Definition at line 361 of file rdata.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.8 initField1()

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

initialise vector and attach to the field

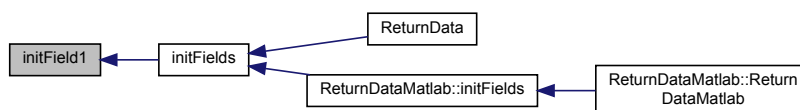
#### Parameters

<i>fieldPointer</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>dim</i>	number of elements in the vector

Reimplemented in [ReturnDataMatlab](#).

Definition at line 441 of file `rdata.cpp`.

Here is the caller graph for this function:



### 9.17.3.9 initField2()

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

initialise matrix and attach to the field

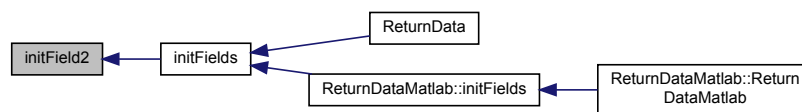
**Parameters**

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

Reimplemented in [ReturnDataMatlab](#).

Definition at line 452 of file rdata.cpp.

Here is the caller graph for this function:

**9.17.3.10 initField3()**

```

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

```

initialise 3D tensor and attach to the field

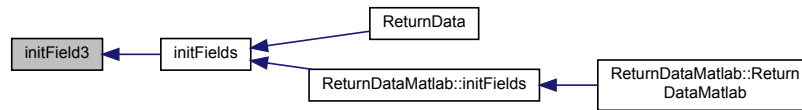
**Parameters**

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

Reimplemented in [ReturnDataMatlab](#).

Definition at line 464 of file rdata.cpp.

Here is the caller graph for this function:



### 9.17.3.11 initField4()

```

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

initialise 4D tensor and attach to the field

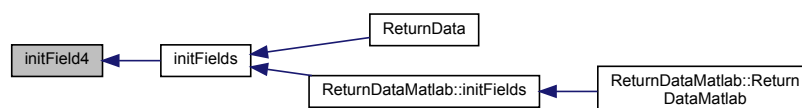
#### Parameters

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

Reimplemented in [ReturnDataMatlab](#).

Definition at line 478 of file `rdata.cpp`.

Here is the caller graph for this function:



### 9.17.4 Member Data Documentation

**9.17.4.1 ts**

`double* ts`

timepoints (dimension: nt)

Definition at line 33 of file rdata.h.

**9.17.4.2 xdot**

`double* xdot`

time derivative (dimension: nx)

Definition at line 36 of file rdata.h.

**9.17.4.3 dxdotdp**

`double* dxdotdp`

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

Definition at line 40 of file rdata.h.

**9.17.4.4 dydx**

`double* dydx`

state derivative of observables (dimension: ny x nx, column-major)

Definition at line 43 of file rdata.h.

**9.17.4.5 dydp**

`double* dydp`

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

Definition at line 47 of file rdata.h.



#### 9.17.4.6 J

double\* J

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

Definition at line 51 of file rdata.h.

#### 9.17.4.7 z

double\* z

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

Definition at line 54 of file rdata.h.

#### 9.17.4.8 sigmaz

double\* sigmaz

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

Definition at line 58 of file rdata.h.

#### 9.17.4.9 sz

double\* sz

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

Definition at line 62 of file rdata.h.

#### 9.17.4.10 ssigmaz

double\* ssigmaz

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

Definition at line 66 of file rdata.h.

**9.17.4.11 rz**

`double* rz`

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

Definition at line 69 of file rdata.h.

**9.17.4.12 srz**

`double* srz`

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

Definition at line 73 of file rdata.h.

**9.17.4.13 s2rz**

`double* s2rz`

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

Definition at line 77 of file rdata.h.

**9.17.4.14 x**

`double* x`

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

Definition at line 80 of file rdata.h.

**9.17.4.15 sx**

`double* sx`

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

Definition at line 84 of file rdata.h.

**9.17.4.16 y**

`double* y`

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

Definition at line 87 of file rdata.h.

**9.17.4.17 sigmay**

`double* sigmay`

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

Definition at line 90 of file rdata.h.

**9.17.4.18 sy**

`double* sy`

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

Definition at line 94 of file rdata.h.

**9.17.4.19 ssigmay**

`double* ssigmay`

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

Definition at line 98 of file rdata.h.

**9.17.4.20 numsteps**

`double* numsteps`

number of integration steps forward problem (dimension: nt)

Definition at line 101 of file rdata.h.

**9.17.4.21 numstepsB**

`double* numstepsB`

number of integration steps backward problem (dimension: nt)

Definition at line 104 of file rdata.h.

**9.17.4.22 numrhsevals**

`double* numrhsevals`

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

Definition at line 107 of file rdata.h.

**9.17.4.23 numrhsevalsB**

`double* numrhsevalsB`

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

Definition at line 110 of file rdata.h.

**9.17.4.24 numerrtestfails**

`double* numerrtestfails`

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

Definition at line 113 of file rdata.h.

**9.17.4.25 numerrtestfailsB**

`double* numerrtestfailsB`

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

Definition at line 116 of file rdata.h.

**9.17.4.26 numnonlinsolvconvfails**

```
double* numnonlinsolvconvfails
```

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

Definition at line 120 of file rdata.h.

**9.17.4.27 numnonlinsolvconvfailsB**

```
double* numnonlinsolvconvfailsB
```

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

Definition at line 124 of file rdata.h.

**9.17.4.28 order**

```
double* order
```

employed order forward problem (dimension: nt)

Definition at line 127 of file rdata.h.

**9.17.4.29 newton\_status**

```
double* newton_status
```

flag indicating success of Newton solver

Definition at line 130 of file rdata.h.

**9.17.4.30 newton\_time**

```
double* newton_time
```

computation time of the Newton solver [s]

Definition at line 133 of file rdata.h.

**9.17.4.31 newton\_numsteps**

```
double* newton_numsteps
```

number of Newton steps for steady state problem

Definition at line 136 of file rdata.h.

**9.17.4.32 newton\_numlinsteps**

```
double* newton_numlinsteps
```

number of linear steps by Newton step for steady state problem

Definition at line 139 of file rdata.h.

**9.17.4.33 xss**

```
double* xss
```

steady state found be Newton solver

Definition at line 142 of file rdata.h.

**9.17.4.34 llh**

```
double* llh
```

likelihood value (double[1])

Definition at line 145 of file rdata.h.

**9.17.4.35 chi2**

```
double* chi2
```

chi2 value (double[1])

Definition at line 148 of file rdata.h.

**9.17.4.36 sllh**

```
double* sllh
```

parameter derivative of likelihood (dimension: nplist)

Definition at line 151 of file rdata.h.

**9.17.4.37 s2llh**

```
double* s2llh
```

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

Definition at line 155 of file rdata.h.

**9.17.4.38 status**

```
double* status
```

status code (double[1])

Definition at line 158 of file rdata.h.

**9.17.4.39 freeFieldsOnDestruction**

```
bool freeFieldsOnDestruction [protected]
```

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

Definition at line 178 of file rdata.h.

**9.17.4.40 np**

```
const int np
```

total number of model parameters

Definition at line 182 of file rdata.h.

**9.17.4.41 nk**

```
const int nk
```

number of fixed parameters

Definition at line 184 of file rdata.h.

**9.17.4.42 nx**

```
const int nx
```

number of states

Definition at line 186 of file rdata.h.

**9.17.4.43 nxtrue**

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 188 of file rdata.h.

**9.17.4.44 ny**

```
const int ny
```

number of observables

Definition at line 190 of file rdata.h.

**9.17.4.45 nytrue**

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 192 of file rdata.h.



**9.17.4.46 nz**

```
const int nz
```

number of event outputs

Definition at line 194 of file rdata.h.

**9.17.4.47 nztrue**

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 196 of file rdata.h.

**9.17.4.48 ne**

```
const int ne
```

number of events

Definition at line 198 of file rdata.h.

**9.17.4.49 nJ**

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 200 of file rdata.h.

**9.17.4.50 nplist**

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 203 of file rdata.h.

**9.17.4.51 nmaxevent**

```
const int nmaxevent
```

maximal number of occuring events (for every event type)

Definition at line 205 of file rdata.h.

**9.17.4.52 nt**

```
const int nt
```

number of considered timepoints

Definition at line 207 of file rdata.h.

**9.17.4.53 newton\_maxsteps**

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 209 of file rdata.h.

**9.17.4.54 pscale**

```
const AMICI_parameter_scaling pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 211 of file rdata.h.

**9.17.4.55 o2mode**

```
const AMICI_o2mode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 213 of file rdata.h.

## 9.17.4.56 sensi

```
const AMICI_sensi_order sensi
```

sensitivity order

Definition at line 215 of file rdata.h.

## 9.17.4.57 sensi\_meth

```
const AMICI_sensi_meth sensi_meth
```

sensitivity method

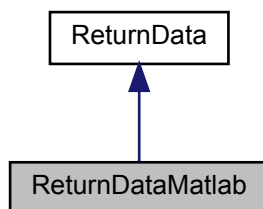
Definition at line 217 of file rdata.h.

## 9.18 ReturnDataMatlab Class Reference

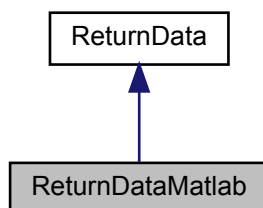
The [ReturnDataMatlab](#) class sets up [ReturnData](#) to be returned by the MATLAB mex functions. Memory is allocated using matlab functions.

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

Inheritance diagram for ReturnDataMatlab:



Collaboration diagram for ReturnDataMatlab:



## Public Member Functions

- [ReturnDataMatlab](#) (const [UserData](#) \*udata, const [Model](#) \*model)

## Public Attributes

- mxArray \* [mxsol](#)

## Protected Member Functions

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

## Additional Inherited Members

### 9.18.1 Detailed Description

Definition at line 15 of file returndata\_matlab.h.

### 9.18.2 Constructor & Destructor Documentation

#### 9.18.2.1 ReturnDataMatlab()

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

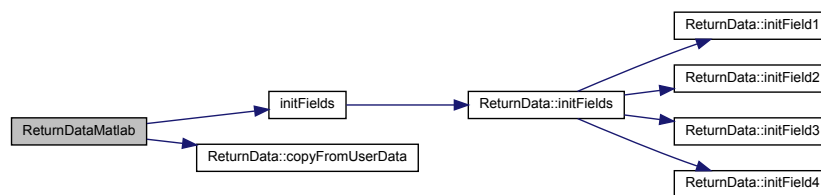
initialises the returnData struct, initialises the fields and copies model dimensions from the udata struct

#### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

Definition at line 3 of file returndata\_matlab.cpp.

Here is the call graph for this function:



### 9.18.3 Member Function Documentation

#### 9.18.3.1 initFields()

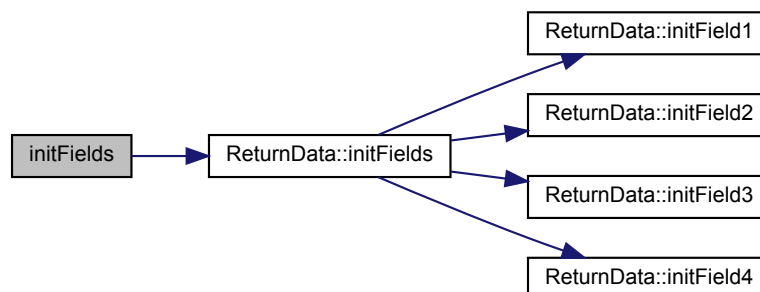
```
void initFields ( ) [protected], [virtual]
```

initialises sol object with the corresponding fields

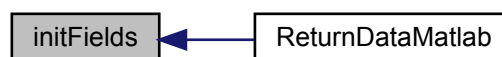
Reimplemented from [ReturnData](#).

Definition at line 17 of file `returndata_matlab.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.18.3.2 initField1()

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

initialise vector and attach to the field

#### Parameters

<i>fieldPointer</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>dim</i>	number of elements in the vector

Reimplemented from [ReturnData](#).

Definition at line 66 of file returndata\_matlab.cpp.

### 9.18.3.3 initField2()

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

initialise matrix and attach to the field

#### Parameters

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

Reimplemented from [ReturnData](#).

Definition at line 83 of file returndata\_matlab.cpp.

### 9.18.3.4 initField3()

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

initialise 3D tensor and attach to the field

## Parameters

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

Reimplemented from [ReturnData](#).

Definition at line 101 of file returndata\_matlab.cpp.

## 9.18.3.5 initField4()

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

initialise 4D tensor and attach to the field

## Parameters

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

Reimplemented from [ReturnData](#).

Definition at line 121 of file returndata\_matlab.cpp.

## 9.18.4 Member Data Documentation

## 9.18.4.1 mxsol

```
mxArray* mxsol
```

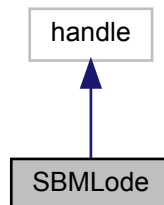
sol struct that is passed back to matlab

Definition at line 22 of file returndata\_matlab.h.

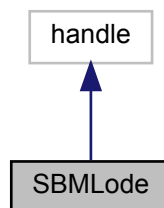
## 9.19 SBMLode Class Reference

SBMLMODEL provides an intermediate container between the SBML definition and an amimodel object.

Inheritance diagram for SBMLode:



Collaboration diagram for SBMLode:



### Public Member Functions

- `SBMLode` (matlabtypesubstitute filename)  
*`SBMLode` extracts information from an SBML definition and stores it in a symbolic format.*
- `noret::substitute checkODE` (matlabtypesubstitute filename, matlabtypesubstitute this)  
*`checkODE` checks whether the length of various variable names exceeds `namelengthmax` (would cause trouble with symbolic processing later on).*
- `noret::substitute writeAMICI` (matlabtypesubstitute modelname)  
*`writeAMICI` writes the symbolic information from an `SBMLode` object into an AMICI model definition file*

### Public Attributes

- matlabtypesubstitute `state` = `sym.empty("")`  
*states*
- matlabtypesubstitute `observable` = `sym.empty("")`



- observables*
- matlabtypesubstitute [observable\\_name](#) = sym.empty("")
- names of observables*
- matlabtypesubstitute [param](#) = sym.empty("")
- parameter names*
- matlabtypesubstitute [parameter](#) = sym.empty("")
- parameter expressions*
- matlabtypesubstitute [constant](#) = sym.empty("")
- constants*
- matlabtypesubstitute [reaction](#) = sym.empty("")
- reactions*
- matlabtypesubstitute [compartment](#) = sym.empty("")
- compartments*
- matlabtypesubstitute [volume](#) = sym.empty("")
- compartment volumes*
- matlabtypesubstitute [kvolume](#) = sym.empty("")
- condition volumes*
- matlabtypesubstitute [initState](#) = sym.empty("")
- initial condition of states*
- matlabtypesubstitute [condition](#) = sym.empty("")
- condition*
- matlabtypesubstitute [flux](#) = sym.empty("")
- reaction fluxes*
- matlabtypesubstitute [stoichiometry](#) = sym.empty("")
- reaction stoichiometry*
- matlabtypesubstitute [xdot](#) = sym.empty("")
- right hand side of reconstructed differential equation*
- matlabtypesubstitute [trigger](#) = sym.empty("")
- event triggers*
- matlabtypesubstitute [bolus](#) = sym.empty("")
- event boli*
- matlabtypesubstitute [funmath](#) = cell.empty("")
- mathematical expressions for function*
- matlabtypesubstitute [funarg](#) = cell.empty("")
- matlabtypesubstitute [time\\_symbol](#) = char.empty("")
- symbol of time*
- matlabtypesubstitute [pnom](#) = double.empty("")
- nominal parameters*
- matlabtypesubstitute [knom](#) = double.empty("")
- nominal conditions*

### 9.19.1 Detailed Description

Definition at line 17 of file SBMLode.m.

### 9.19.2 Constructor & Destructor Documentation

#### 9.19.2.1 SBMLode()

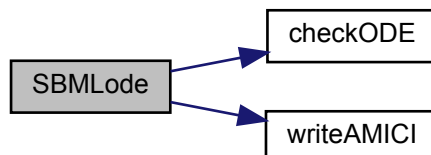
```
SBMLode (
    matlabtypesubstitute filename )
```

**Parameters**

<i>filename</i>	target name of the model (excluding the suffix .xml/.sbml)
-----------------	--

Definition at line 201 of file SBMLode.m.

Here is the call graph for this function:

**9.19.3 Member Function Documentation****9.19.3.1 checkODE()**

```

noret::substitute checkODE (
    matlabtypesubstitute filename,
    matlabtypesubstitute this )
  
```

**Parameters**

<i>this</i>	<code>SBMLode</code> object
-------------	-----------------------------

**Return values**

<i>this</i>	void
-------------	------

Definition at line 18 of file checkODE.m.

Here is the caller graph for this function:



### 9.19.3.2 writeAMICI()

```
noret::substitute writeAMICI (  
    matlabtypesubstitute modelName )
```

#### Parameters

<i>modelName</i>	target name of the model (_syms.m will be appended to the name )
------------------	--

Definition at line 18 of file writeAMICI.m.

Here is the caller graph for this function:



## 9.19.4 Member Data Documentation

### 9.19.4.1 state

```
state = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 28 of file SBMLode.m.

### 9.19.4.2 observable

```
observable = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 36 of file SBMLode.m.

#### 9.19.4.3 observable\_name

```
observable_name = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 44 of file SBMLode.m.

#### 9.19.4.4 param

```
param = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 52 of file SBMLode.m.

#### 9.19.4.5 parameter

```
parameter = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 60 of file SBMLode.m.

#### 9.19.4.6 constant

```
constant = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 68 of file SBMLode.m.

#### 9.19.4.7 reaction

```
reaction = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 76 of file SBMLode.m.

#### 9.19.4.8 compartment

```
compartment = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 84 of file SBMLode.m.

#### 9.19.4.9 volume

```
volume = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 92 of file SBMLode.m.

#### 9.19.4.10 kvolume

```
kvolume = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 100 of file SBMLode.m.

#### 9.19.4.11 initState

```
initState = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 108 of file SBMLode.m.

#### 9.19.4.12 condition

```
condition = sym.empty("")
```

**Default:** sym.empty("")

Definition at line 116 of file SBMLode.m.

#### 9.19.4.13 flux

```
flux = sym.empty("")
```

**Default:** `sym.empty("")`

Definition at line 124 of file SBMLode.m.

#### 9.19.4.14 stoichiometry

```
stoichiometry = sym.empty("")
```

**Default:** `sym.empty("")`

Definition at line 132 of file SBMLode.m.

#### 9.19.4.15 xdot

```
xdot = sym.empty("")
```

**Default:** `sym.empty("")`

Definition at line 140 of file SBMLode.m.

#### 9.19.4.16 trigger

```
trigger = sym.empty("")
```

**Default:** `sym.empty("")`

Definition at line 148 of file SBMLode.m.

#### 9.19.4.17 bolus

```
bolus = sym.empty("")
```

**Default:** `sym.empty("")`

Definition at line 156 of file SBMLode.m.

#### 9.19.4.18 funmath

```
funmath = cell.empty("")
```

**Default:** cell.empty("")

Definition at line 164 of file SBMLode.m.

#### 9.19.4.19 time\_symbol

```
time_symbol = char.empty("")
```

**Default:** char.empty("")

Definition at line 174 of file SBMLode.m.

#### 9.19.4.20 pnom

```
pnom = double.empty("")
```

**Default:** double.empty("")

Definition at line 182 of file SBMLode.m.

#### 9.19.4.21 knom

```
knom = double.empty("")
```

**Default:** double.empty("")

Definition at line 190 of file SBMLode.m.

## 9.20 Solver Class Reference

[Solver](#) class.

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

## Public Member Functions

- int **setupAMI** (UserData \*udata, TempData \*tdata, Model \*model)  
*setupAMIs initialises the ami memory object*
- int **setupAMIB** (UserData \*udata, TempData \*tdata, Model \*model)
- virtual int **AMIGetSens** (realtype \*tret, N\_Vector \*yySout)=0
- int **getDiagnosis** (int it, ReturnData \*rdata)
- int **getDiagnosisB** (int it, ReturnData \*rdata, TempData \*tdata)
- virtual int **AMIGetRootInfo** (int \*rootsfound)=0
- virtual int **AMIRelnit** (realtype t0, N\_Vector yy0, N\_Vector yp0)=0
- virtual int **AMISensRelnit** (int ism, N\_Vector \*yS0, N\_Vector \*ypS0)=0
- virtual int **AMICalcIC** (realtype tout1)=0
- virtual int **AMICalcICB** (int which, realtype tout1, N\_Vector xB, N\_Vector dxB)=0
- virtual int **AMISolve** (realtype tout, N\_Vector yret, N\_Vector ypret, realtype \*tret, int itask)=0
- virtual int **AMISolveF** (realtype tout, N\_Vector yret, N\_Vector ypret, realtype \*tret, int itask, int \*ncheckPtr)=0
- virtual int **AMISolveB** (realtype tBout, int itaskB)=0
- virtual int **AMISetStopTime** (realtype tstop)=0
- virtual int **AMIRelnitB** (int which, realtype tB0, N\_Vector yyB0, N\_Vector ypB0)=0
- virtual int **AMIGetB** (int which, realtype \*tret, N\_Vector yy, N\_Vector yp)=0
- virtual int **AMIGetQuadB** (int which, realtype \*tret, N\_Vector qB)=0
- virtual int **AMIQuadRelnitB** (int which, N\_Vector yQB0)=0
- virtual int **turnOffRootFinding** ()=0

## Protected Member Functions

- virtual int **wrap\_init** (N\_Vector x, N\_Vector dx, realtype t)=0
- virtual int **wrap\_binit** (int which, N\_Vector xB, N\_Vector dxB, realtype t)=0
- virtual int **wrap\_qbinit** (int which, N\_Vector qBdot)=0
- virtual int **wrap\_RootInit** (int ne)=0
- virtual int **wrap\_SensInit1** (N\_Vector \*sx, N\_Vector \*sdx, UserData \*udata)=0
- virtual int **wrap\_SetDenseJacFn** ()=0
- virtual int **wrap\_SetSparseJacFn** ()=0
- virtual int **wrap\_SetBandJacFn** ()=0
- virtual int **wrap\_SetJacTimesVecFn** ()=0
- virtual int **wrap\_SetDenseJacFnB** (int which)=0
- virtual int **wrap\_SetSparseJacFnB** (int which)=0
- virtual int **wrap\_SetBandJacFnB** (int which)=0
- virtual int **wrap\_SetJacTimesVecFnB** (int which)=0
- virtual void \* **AMICreate** (int lmm, int iter)=0
- virtual int **AMISStolerances** (double rtol, double atol)=0
- virtual int **AMISensEEtolerances** ()=0
- virtual int **AMISetSensErrCon** (bool error\_corr)=0
- virtual int **AMISetQuadErrConB** (int which, bool flag)=0
- virtual int **AMISetErrHandlerFn** ()=0
- virtual int **AMISetUserData** (void \*user\_data)=0
- virtual int **AMISetUserDataB** (int which, void \*user\_data)=0
- virtual int **AMISetMaxNumSteps** (long int mxsteps)=0
- virtual int **AMISetStabLimDet** (int stldet)=0
- virtual int **AMISetStabLimDetB** (int which, int stldet)=0
- virtual int **AMISetId** (Model \*model)=0
- virtual int **AMISetSuppressAlg** (bool flag)=0
- virtual int **AMISetSensParams** (realtype \*p, realtype \*pbar, int \*plist)=0
- virtual int **AMIGetDky** (realtype t, int k, N\_Vector dky)=0
- virtual void **AMIFree** ()=0



- virtual int **AMIAdjInit** (long int steps, int interp)=0
- virtual int **AMICreateB** (int lmm, int iter, int \*which)=0
- virtual int **AMISStolerancesB** (int which, realtype relTolB, realtype absTolB)=0
- virtual int **AMIQuadStolerancesB** (int which, realtype reltolQB, realtype abstolQB)=0
- virtual int **AMISetMaxNumStepsB** (int which, long int mxstepsB)=0
- virtual int **AMIDense** (int nx)=0
- virtual int **AMIDenseB** (int which, int nx)=0
- virtual int **AMIBand** (int nx, int ubw, int lbw)=0
- virtual int **AMIBandB** (int which, int nx, int ubw, int lbw)=0
- virtual int **AMIDiag** ()=0
- virtual int **AMIDiagB** (int which)=0
- virtual int **AMISpgmr** (int prectype, int maxl)=0
- virtual int **AMISpgmrB** (int which, int prectype, int maxl)=0
- virtual int **AMISpbcg** (int prectype, int maxl)=0
- virtual int **AMISpbcgB** (int which, int prectype, int maxl)=0
- virtual int **AMISptfqmr** (int prectype, int maxl)=0
- virtual int **AMISptfqmrB** (int which, int prectype, int maxl)=0
- virtual int **AMIKLU** (int nx, int nnz, int sparsetype)=0
- virtual int **AMIKLUSetOrdering** (int ordering)=0
- virtual int **AMIKLUSetOrderingB** (int which, int ordering)=0
- virtual int **AMIKLUB** (int which, int nx, int nnz, int sparsetype)=0
- virtual int **AMIGetNumSteps** (void \*ami\_mem, long int \*numsteps)=0
- virtual int **AMIGetNumRhsEvals** (void \*ami\_mem, long int \*numrhsevals)=0
- virtual int **AMIGetNumErrTestFails** (void \*ami\_mem, long int \*numerrtestfails)=0
- virtual int **AMIGetNumNonlinSolvConvFails** (void \*ami\_mem, long int \*numnonlinsolvconvfails)=0
- virtual int **AMIGetLastOrder** (void \*ami\_mem, int \*order)=0
- virtual void \* **AMIGetAdjBmem** (void \*ami\_mem, int which)=0
- int **setLinearSolver** (const [UserData](#) \*udata, [Model](#) \*model)

#### Static Protected Member Functions

- static void **wrap\_ErrHandlerFn** (int error\_code, const char \*module, const char \*function, char \*msg, void \*eh\_data)

#### Protected Attributes

- void \* [ami\\_mem](#) = nullptr

### 9.20.1 Detailed Description

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

Definition at line 16 of file amici\_solver.h.

### 9.20.2 Member Function Documentation

#### 9.20.2.1 setupAMI()

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

**Parameters**

out	<i>status</i>	flag indicating success of execution <b>Type:</b> int
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

**Returns**

ami\_mem pointer to the cvodes/idas memory block

Definition at line 15 of file amici\_solver.cpp.

Here is the caller graph for this function:

**9.20.2.2 setupAMIB()**

```

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

```

setupAMIB initialises the AMI memory object for the backwards problem

**Parameters**

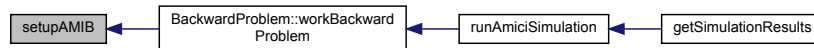
out	<i>status</i>	flag indicating success of execution <b>Type:</b> int
in	<i>ami_mem</i>	pointer to the solver memory object of the forward problem
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

**Returns**

status flag indicating success of execution  
**Type:** int

Definition at line 139 of file amici\_solver.cpp.

Here is the caller graph for this function:



### 9.20.2.3 getDiagnosis()

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

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

#### Parameters

out		
-----	--	--

Definition at line 378 of file amici\_solver.cpp.

Here is the caller graph for this function:



### 9.20.2.4 getDiagnosisB()

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

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

#### Parameters

in	<i>it</i>	time-point index <b>Type:</b> int
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> *void

## Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>

## Returns

status flag indicating success of execution  
**Type:** int

Definition at line 411 of file amici\_solver.cpp.

Here is the caller graph for this function:



## 9.20.3 Member Data Documentation

## 9.20.3.1 ami\_mem

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

pointer to ami memory block

Definition at line 244 of file amici\_solver.h.

## 9.21 SteadystateProblem Class Reference

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

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

## Static Public Member Functions

- static int [workSteadyStateProblem](#) ([UserData](#) \*udata, [TempData](#) \*tdata, [ReturnData](#) \*rdata, int it, [Solver](#) \*solver, [Model](#) \*model)
- static int [applyNewtonsMethod](#) ([UserData](#) \*udata, [ReturnData](#) \*rdata, [TempData](#) \*tdata, int newton\_try, [Model](#) \*model, [NewtonSolver](#) \*newtonSolver)
- static int [getNewtonOutput](#) ([TempData](#) \*tdata, [ReturnData](#) \*rdata, int newton\_status, double run\_time, int nx)
- static int [getNewtonSimulation](#) ([UserData](#) \*udata, [TempData](#) \*tdata, [ReturnData](#) \*rdata, [Solver](#) \*solver, [Model](#) \*model)
- static int [linSolveSPBCG](#) ([UserData](#) \*udata, [ReturnData](#) \*rdata, [TempData](#) \*tdata, int ntry, int nnewt, N\_Vector ns\_delta, [Model](#) \*model)

### 9.21.1 Detailed Description

Definition at line 21 of file steadystateproblem.h.

### 9.21.2 Member Function Documentation

#### 9.21.2.1 workSteadyStateProblem()

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

tries to determine the steady state of the ODE system by a Newton

solver uses forward intergration, if the Newton solver fails

#### Parameters

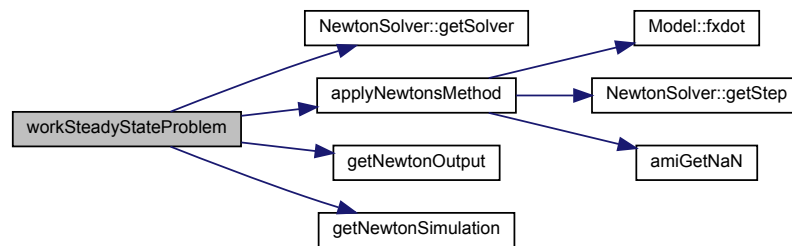
in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>

#### Returns

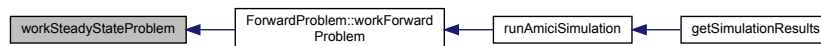
status flag indicating success of execution  
**Type:** int

Definition at line 16 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.21.2.2 applyNewtonsMethod()

```

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

applyNewtonsMethod applies Newtons method to the current state x to find the steady state

applyNewtonsMethod applies Newtons method to the current state x to

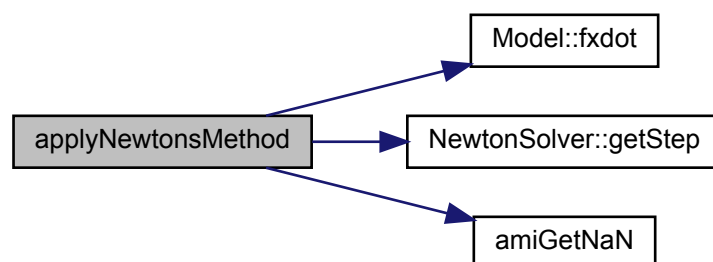
find the steady state

```

@param[in] udata pointer to the user data struct <br><b>Type</b>: UserData
@param[in] tdata pointer to the temporary data struct <br><b>Type</b>: TempData
@param[out] tdata pointer to the temporary data struct <br><b>Type</b>: TempData
@param[out] rdata pointer to the return data struct <br><b>Type</b>: ReturnData
@param[in] newton_try integer for the try number of the Newton solver
@return status flag indicating success of execution <br><b>Type</b>: int
  
```

Definition at line 84 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.21.2.3 getNewtonOutput()

```

int getNewtonOutput (
    TempData * tdata,
    ReturnData * rdata,
    int newton_status,
    double run_time,
    int nx ) [static]

```

`getNewtonOutput` stores the output of the Newton solver run.

@param[in] udata pointer to the user data struct <br><b>Type</b>: UserData  
 @param[in] tdata pointer to the temporary data struct <br><b>Type</b>: TempData  
 @param[out] rdata pointer to the return data struct <br><b>Type</b>: ReturnData  
 @param[in] newton\_status integer flag indicating the run of the

Newton solver

#### Parameters

in	<i>run_time</i>	double computation time of the Newton solver
----	-----------------	--

**Returns**

status flag indicating success of execution

**Type:** int

Definition at line 238 of file steadystateproblem.cpp.

Here is the caller graph for this function:

**9.21.2.4 getNewtonSimulation()**

```

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

getNewtonSimulation solves the forward problem, if the first Newton

solver run did not succeed.

@param[in] udata pointer to the user data struct <br><b>Type</b>: UserData  
 @param[in] tdata pointer to the temporary data struct <br><b>Type</b>: TempData  
 @param[in] rdata pointer to the return data struct <br><b>Type</b>: ReturnData  
 @param[out] rdata pointer to the return data struct <br><b>Type</b>: ReturnData  
 @return status flag indicating success of execution <br><b>Type</b>: int

Definition at line 280 of file steadystateproblem.cpp.

Here is the caller graph for this function:

**9.21.2.5 linsolveSPBCG()**

```

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

linsolveSPBCG solves the linear system for the Newton iteration by

using the BiCGStab algorithm. This routines is to be stored in another file in near future.



## Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
out	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>
in	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
out	<i>tdata</i>	pointer to the temporary data struct <b>Type:</b> <a href="#">TempData</a>
in	<i>ami_mem</i>	pointer to the solver memory block <b>Type:</b> *void
in	<i>ntry</i>	intger number of Newton solver try
in	<i>nnewt</i>	intger number of Newton steps in the current Newton solver try
out	<i>delta</i>	N_Vector solution of the linear system

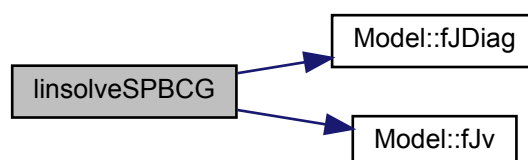
## Returns

status flag indicating success of execution

**Type:** int

Definition at line 347 of file steadystateproblem.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

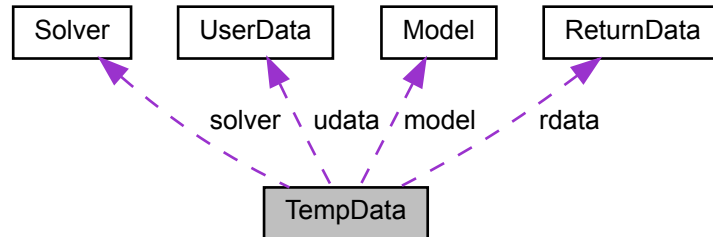


## 9.22 TempData Class Reference

struct that provides temporary storage for different variables

```
#include <tdata.h>
```

Collaboration diagram for TempData:



### Public Member Functions

- [TempData](#) (const [UserData](#) \**udata*, [Model](#) \**model*, [ReturnData](#) \**rdata*)

### Public Attributes

- `realtype * p = nullptr`
- `realtype t`
- `N_Vector x`
- `N_Vector x_old`
- `N_Vector * x_disc`
- `N_Vector * xdot_disc`
- `N_Vector * xdot_old_disc`
- `N_Vector dx`
- `N_Vector dx_old`
- `N_Vector xdot`
- `N_Vector xdot_old`
- `N_Vector xB`
- `N_Vector xB_old`
- `N_Vector dxB`
- `N_Vector xQB`
- `N_Vector xQB_old`
- `N_Vector * sx`
- `N_Vector * sdx`
- `DisMat Jtmp`
- `realtype * llhS0`
- `realtype * Jy`
- `realtype * dJydp`
- `realtype * dJydy`
- `realtype * dJydsigma`

- realtype \* [dJydx](#)
- realtype \* [Jz](#)
- realtype \* [dJzdp](#)
- realtype \* [dJzdx](#)
- realtype \* [dJzdz](#)
- realtype \* [dJzdsigma](#)
- realtype \* [dJrzdz](#)
- realtype \* [dJrzdsigma](#)
- realtype \* [dzdx](#)
- realtype \* [dzdp](#)
- realtype \* [drzdx](#)
- realtype \* [drzdp](#)
- realtype \* [dydp](#)
- realtype \* [dydx](#)
- realtype \* [yS0](#)
- realtype \* [sigmay](#)
- realtype \* [dsigmaydp](#)
- realtype \* [sigmaz](#)
- realtype \* [dsigmazdp](#)
- int \* [rootsfound](#)
- int \* [rootidx](#)
- int \* [nroots](#)
- realtype \* [rootvals](#)
- realtype \* [h](#)
- realtype \* [h\\_adata](#)
- realtype \* [deltax](#)
- realtype \* [deltasx](#)
- realtype \* [deltaxB](#)
- realtype \* [deltaqB](#)
- int [which](#)
- realtype \* [discs](#)
- realtype \* [irdiscs](#)
- SlsMat [J](#) = NULL
- realtype \* [dxdotdp](#) = NULL
- realtype \* [w](#) = NULL
- realtype \* [dwdx](#) = NULL
- realtype \* [dwdp](#) = NULL
- realtype \* [M](#) = NULL
- realtype \* [dfdx](#) = NULL
- realtype \* [stau](#) = NULL
- int [nplist](#)
- int [iroot](#) = 0
- booleantype [nan\\_dxdotdp](#) = false
- booleantype [nan\\_J](#) = false
- booleantype [nan\\_JDiag](#) = false
- booleantype [nan\\_JSparse](#) = false
- booleantype [nan\\_xdot](#) = false
- booleantype [nan\\_xBdot](#) = false
- booleantype [nan\\_qBdot](#) = false
- const [UserData](#) \* [udata](#)
- [Model](#) \* [model](#)
- [ReturnData](#) \* [rdata](#)
- [Solver](#) \* [solver](#) = nullptr

### 9.22.1 Detailed Description

Definition at line 17 of file tdata.h.

### 9.22.2 Constructor & Destructor Documentation

#### 9.22.2.1 TempData()

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

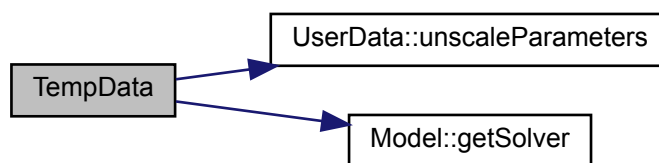
Default constructor

#### Parameters

in	<i>udata</i>	pointer to the user data struct <b>Type:</b> <a href="#">UserData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>
in	<i>rdata</i>	pointer to the return data struct <b>Type:</b> <a href="#">ReturnData</a>

Definition at line 8 of file tdata.cpp.

Here is the call graph for this function:



### 9.22.3 Member Data Documentation

#### 9.22.3.1 p

```
realtype* p = nullptr
parameter array, unscaled
```

Definition at line 24 of file tdata.h.

### 9.22.3.2 t

`realtype t`

current time

Definition at line 27 of file tdata.h.

### 9.22.3.3 x

`N_Vector x`

state vector

Definition at line 30 of file tdata.h.

### 9.22.3.4 x\_old

`N_Vector x_old`

old state vector

Definition at line 32 of file tdata.h.

### 9.22.3.5 x\_disc

`N_Vector* x_disc`

array of state vectors at discontinuities

Definition at line 34 of file tdata.h.

### 9.22.3.6 xdot\_disc

`N_Vector* xdot_disc`

array of differential state vectors at discontinuities

Definition at line 36 of file tdata.h.

#### 9.22.3.7 `xdot_old_disc`

`N_Vector* xdot_old_disc`

array of old differential state vectors at discontinuities

Definition at line 38 of file tdata.h.

#### 9.22.3.8 `dx`

`N_Vector dx`

differential state vector

Definition at line 40 of file tdata.h.

#### 9.22.3.9 `dx_old`

`N_Vector dx_old`

old differential state vector

Definition at line 42 of file tdata.h.

#### 9.22.3.10 `xdot`

`N_Vector xdot`

time derivative state vector

Definition at line 44 of file tdata.h.

#### 9.22.3.11 `xdot_old`

`N_Vector xdot_old`

old time derivative state vector

Definition at line 46 of file tdata.h.

**9.22.3.12 xB**

`N_Vector xB`

adjoint state vector

Definition at line 48 of file tdata.h.

**9.22.3.13 xB\_old**

`N_Vector xB_old`

old adjoint state vector

Definition at line 50 of file tdata.h.

**9.22.3.14 dxB**

`N_Vector dxB`

differential adjoint state vector

Definition at line 52 of file tdata.h.

**9.22.3.15 xQB**

`N_Vector xQB`

quadrature state vector

Definition at line 54 of file tdata.h.

**9.22.3.16 xQB\_old**

`N_Vector xQB_old`

old quadrature state vector

Definition at line 56 of file tdata.h.

**9.22.3.17 sx**

```
N_Vector* sx
```

sensitivity state vector array

Definition at line 58 of file tdata.h.

**9.22.3.18 sdx**

```
N_Vector* sdx
```

differential sensitivity state vector array

Definition at line 60 of file tdata.h.

**9.22.3.19 Jtmp**

```
DlsMat Jtmp
```

Jacobian

Definition at line 62 of file tdata.h.

**9.22.3.20 llhS0**

```
realtype* llhS0
```

parameter derivative of likelihood array

Definition at line 65 of file tdata.h.

**9.22.3.21 Jy**

```
realtype* Jy
```

data likelihood

Definition at line 67 of file tdata.h.



#### 9.22.3.22 dJydp

`realtype* dJydp`

parameter derivative of data likelihood

Definition at line 69 of file tdata.h.

#### 9.22.3.23 dJydy

`realtype* dJydy`

observable derivative of data likelihood

Definition at line 71 of file tdata.h.

#### 9.22.3.24 dJydsigma

`realtype* dJydsigma`

observable sigma derivative of data likelihood

Definition at line 73 of file tdata.h.

#### 9.22.3.25 dJydx

`realtype* dJydx`

state derivative of data likelihood

Definition at line 75 of file tdata.h.

#### 9.22.3.26 Jz

`realtype* Jz`

event likelihood

Definition at line 77 of file tdata.h.

**9.22.3.27 dJzdp**

`realtype* dJzdp`

parameter derivative of event likelihood

Definition at line 79 of file tdata.h.

**9.22.3.28 dJzdx**

`realtype* dJzdx`

state derivative of event likelihood

Definition at line 81 of file tdata.h.

**9.22.3.29 dJzdz**

`realtype* dJzdz`

event output derivative of event likelihood

Definition at line 83 of file tdata.h.

**9.22.3.30 dJzdsigma**

`realtype* dJzdsigma`

event sigma derivative of event likelihood

Definition at line 85 of file tdata.h.

**9.22.3.31 dJrzdz**

`realtype* dJrzdz`

event output derivative of event likelihood at final timepoint

Definition at line 87 of file tdata.h.

### 9.22.3.32 dJrzdsigma

`realtype* dJrzdsigma`

event sigma derivative of event likelihood at final timepoint

Definition at line 89 of file tdata.h.

### 9.22.3.33 dzdx

`realtype* dzdx`

state derivative of event output

Definition at line 91 of file tdata.h.

### 9.22.3.34 dzdp

`realtype* dzdp`

parameter derivative of event output

Definition at line 93 of file tdata.h.

### 9.22.3.35 drzdx

`realtype* drzdx`

state derivative of event timepoint

Definition at line 95 of file tdata.h.

### 9.22.3.36 drzdp

`realtype* drzdp`

parameter derivative of event timepoint

Definition at line 97 of file tdata.h.

**9.22.3.37 dydp**

`realtype* dydp`

parameter derivative of observable

Definition at line 99 of file tdata.h.

**9.22.3.38 dydx**

`realtype* dydx`

state derivative of observable

Definition at line 101 of file tdata.h.

**9.22.3.39 yS0**

`realtype* yS0`

initial sensitivity of observable

Definition at line 103 of file tdata.h.

**9.22.3.40 sigmay**

`realtype* sigmay`

data standard deviation

Definition at line 105 of file tdata.h.

**9.22.3.41 dsigmaydp**

`realtype* dsigmaydp`

parameter derivative of data standard deviation

Definition at line 107 of file tdata.h.

**9.22.3.42 sigmaz**

```
realtype* sigmaz
```

event standard deviation

Definition at line 109 of file tdata.h.

**9.22.3.43 dsigmazdp**

```
realtype* dsigmazdp
```

parameter derivative of event standard deviation

Definition at line 111 of file tdata.h.

**9.22.3.44 rootsfound**

```
int* rootsfound
```

array of flags indicating which root has beend found

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

Definition at line 118 of file tdata.h.

**9.22.3.45 rootidx**

```
int* rootidx
```

array of index which root has been found

Definition at line 120 of file tdata.h.

**9.22.3.46 nroots**

```
int* nroots
```

array of number of found roots for a certain event type

Definition at line 122 of file tdata.h.

#### 9.22.3.47 rootvals

`realtype* rootvals`

array of values of the root function

Definition at line 124 of file tdata.h.

#### 9.22.3.48 h

`realtype* h`

temporary rootval storage to check crossing in secondary event

Definition at line 126 of file tdata.h.

#### 9.22.3.49 h\_adata

`realtype* h_adata`

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

Definition at line 131 of file tdata.h.

#### 9.22.3.50 deltax

`realtype* deltax`

change in x

Definition at line 134 of file tdata.h.

#### 9.22.3.51 deltasx

`realtype* deltasx`

change in sx

Definition at line 136 of file tdata.h.

**9.22.3.52** `deltaxB`

```
realtype* deltaxB
```

change in xB

Definition at line 138 of file tdata.h.

**9.22.3.53** `deltaqB`

```
realtype* deltaqB
```

change in qB

Definition at line 140 of file tdata.h.

**9.22.3.54** `which`

```
int which
```

integer for indexing of backwards problems

Definition at line 143 of file tdata.h.

**9.22.3.55** `discs`

```
realtype* discs
```

array containing the time-points of discontinuities

Definition at line 146 of file tdata.h.

**9.22.3.56** `irdiscs`

```
realtype* irdiscs
```

array containing the index of discontinuities

Definition at line 148 of file tdata.h.

**9.22.3.57 J**

```
SlsMat J = NULL
```

temporary storage of Jacobian data across functions

Definition at line 151 of file tdata.h.

**9.22.3.58 dxdotdp**

```
realtype* dxdotdp = NULL
```

temporary storage of dxdotdp data across functions

Definition at line 153 of file tdata.h.

**9.22.3.59 w**

```
realtype* w = NULL
```

temporary storage of w data across functions

Definition at line 155 of file tdata.h.

**9.22.3.60 dwdx**

```
realtype* dwdx = NULL
```

temporary storage of dwdx data across functions

Definition at line 157 of file tdata.h.

**9.22.3.61 dwdp**

```
realtype* dwdp = NULL
```

temporary storage of dwdp data across functions

Definition at line 159 of file tdata.h.



**9.22.3.62 M**

```
realtype* M = NULL
```

temporary storage of M data across functions

Definition at line 161 of file tdata.h.

**9.22.3.63 dfdx**

```
realtype* dfdx = NULL
```

temporary storage of dfdx data across functions

Definition at line 163 of file tdata.h.

**9.22.3.64 stau**

```
realtype* stau = NULL
```

temporary storage of stau data across functions

Definition at line 165 of file tdata.h.

**9.22.3.65 nplist**

```
int nplist
```

number of parameters, copied from udata, necessary for deallocation

Definition at line 168 of file tdata.h.

**9.22.3.66 iroot**

```
int iroot = 0
```

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

Definition at line 172 of file tdata.h.

**9.22.3.67 nan\_dxdotdp**

```
booleantype nan_dxdotdp = false
```

flag indicating whether a NaN in dxdotdp has been reported

Definition at line 175 of file tdata.h.

**9.22.3.68 nan\_J**

```
booleantype nan_J = false
```

flag indicating whether a NaN in J has been reported

Definition at line 177 of file tdata.h.

**9.22.3.69 nan\_JDiag**

```
booleantype nan_JDiag = false
```

flag indicating whether a NaN in JDiag has been reported

Definition at line 179 of file tdata.h.

**9.22.3.70 nan\_JSparse**

```
booleantype nan_JSparse = false
```

flag indicating whether a NaN in JSparse has been reported

Definition at line 181 of file tdata.h.

**9.22.3.71 nan\_xdot**

```
booleantype nan_xdot = false
```

flag indicating whether a NaN in xdot has been reported

Definition at line 183 of file tdata.h.

#### 9.22.3.72 nan\_xBdot

```
bool nan_xBdot = false
```

flag indicating whether a NaN in xBdot has been reported

Definition at line 185 of file tdata.h.

#### 9.22.3.73 nan\_qBdot

```
bool nan_qBdot = false
```

flag indicating whether a NaN in qBdot has been reported

Definition at line 187 of file tdata.h.

#### 9.22.3.74 udata

```
UserData* udata
```

attached [UserData](#) object

Definition at line 190 of file tdata.h.

#### 9.22.3.75 model

```
Model* model
```

attached [Model](#) object

Definition at line 192 of file tdata.h.

#### 9.22.3.76 rdata

```
ReturnData* rdata
```

attached [ReturnData](#) object

Definition at line 194 of file tdata.h.

#### 9.22.3.77 solver

```
Solver* solver = nullptr
```

attached [Solver](#) object

Definition at line 196 of file tdata.h.

## 9.23 UserData Class Reference

struct that stores all user provided data

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

### Public Member Functions

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

### Public Attributes

- int [nmaxevent](#)
- double \* [qpositivex](#)
- int \* [plist](#)
- int [nplist](#)
- int [nt](#)
- double \* [p](#)
- double \* [k](#)
- AMICI\_parameter\_scaling [pscale](#)
- double [tstart](#)
- double \* [ts](#)
- double \* [pbar](#)
- double \* [xbar](#)
- AMICI\_sensi\_order [sensi](#)
- double [atol](#)
- double [rtol](#)
- int [maxsteps](#)
- int [newton\\_maxsteps](#)
- int [newton\\_maxlinsteps](#)
- int [newton\\_preeq](#)
- int [newton\\_precon](#)
- int [ism](#)
- AMICI\_sensi\_meth [sensi\\_meth](#)
- int [linsol](#)
- int [interpType](#)
- int [lmm](#)
- int [iter](#)
- boolean\_type [stldet](#)
- double \* [x0data](#)
- double \* [sx0data](#)
- int [ordering](#)

### Protected Member Functions

- void [init](#) ()

## 9.23.1 Detailed Description

Definition at line 10 of file udata.h.

## 9.23.2 Member Function Documentation

## 9.23.2.1 unscaleParameters()

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

unscaleParameters removes parameter scaling according to the parameter scaling in pscale

## Parameters

in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>
out	<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 8 of file udata.cpp.

Here is the caller graph for this function:



## 9.23.2.2 print()

```
void print ( )
```

function to print the contents of the [UserData](#) object

Definition at line 92 of file udata.cpp.

### 9.23.2.3 init()

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

function to initialize the contents of the [UserData](#) object

Definition at line 58 of file udata.cpp.

Here is the caller graph for this function:



## 9.23.3 Member Data Documentation

### 9.23.3.1 nmaxevent

```
int nmaxevent
```

maximal number of events to track

Definition at line 25 of file udata.h.

### 9.23.3.2 qpositivex

```
double* qpositivex
```

positivity flag

Definition at line 28 of file udata.h.

### 9.23.3.3 plist

```
int* plist
```

parameter selection and reordering

Definition at line 31 of file udata.h.

#### 9.23.3.4 nplist

```
int nplist
```

number of parameters in plist

Definition at line 33 of file udata.h.

#### 9.23.3.5 nt

```
int nt
```

number of timepoints

Definition at line 36 of file udata.h.

#### 9.23.3.6 p

```
double* p
```

parameter array

Definition at line 39 of file udata.h.

#### 9.23.3.7 k

```
double* k
```

constants array

Definition at line 42 of file udata.h.

#### 9.23.3.8 pscale

```
AMICI_parameter_scaling pscale
```

parameter transformation of p

Definition at line 45 of file udata.h.

#### 9.23.3.9 tstart

`double tstart`

starting time

Definition at line 48 of file udata.h.

#### 9.23.3.10 ts

`double* ts`

timepoints

Definition at line 50 of file udata.h.

#### 9.23.3.11 pbar

`double* pbar`

scaling of parameters

Definition at line 53 of file udata.h.

#### 9.23.3.12 xbar

`double* xbar`

scaling of states

Definition at line 55 of file udata.h.

#### 9.23.3.13 sensi

`AMICI_sensi_order sensi`

flag indicating whether sensitivities are supposed to be computed

Definition at line 58 of file udata.h.



**9.23.3.14 atol**

```
double atol
```

absolute tolerances for integration

Definition at line 60 of file udata.h.

**9.23.3.15 rtol**

```
double rtol
```

relative tolerances for integration

Definition at line 62 of file udata.h.

**9.23.3.16 maxsteps**

```
int maxsteps
```

maximum number of allowed integration steps

Definition at line 64 of file udata.h.

**9.23.3.17 newton\_maxsteps**

```
int newton_maxsteps
```

maximum number of allowed Newton steps for steady state computation

Definition at line 67 of file udata.h.

**9.23.3.18 newton\_maxlinsteps**

```
int newton_maxlinsteps
```

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

Definition at line 70 of file udata.h.

#### 9.23.3.19 newton\_preeq

```
int newton_preeq
```

Preequilibration of model via NEwton solver?

Definition at line 72 of file udata.h.

#### 9.23.3.20 newton\_precon

```
int newton_precon
```

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

Definition at line 75 of file udata.h.

#### 9.23.3.21 ism

```
int ism
```

internal sensitivity method

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

Definition at line 82 of file udata.h.

#### 9.23.3.22 sensi\_meth

```
AMICI_sensi_meth sensi_meth
```

method for sensitivity computation

Definition at line 85 of file udata.h.

#### 9.23.3.23 linsol

```
int linsol
```

linear solver specification

Definition at line 88 of file udata.h.

#### 9.23.3.24 interpType

```
int interpType
```

interpolation type

specifies the interpolation type for the forward problem solution which is then used for the backwards problem. can be either CV\_POLYNOMIAL or CV\_HERMITE

Definition at line 96 of file udata.h.

#### 9.23.3.25 lmm

```
int lmm
```

linear multistep method

specifies the linear multistep method and may be one of two possible values: CV\_ADAMS or CV\_BDF.

Definition at line 103 of file udata.h.

#### 9.23.3.26 iter

```
int iter
```

nonlinear solver

specifies the type of nonlinear solver iteration and may be either CV\_NEWTON or CV\_FUNCTIONAL.

Definition at line 110 of file udata.h.

#### 9.23.3.27 stldet

```
booleantype stldet
```

flag controlling stability limit detection

Definition at line 113 of file udata.h.

#### 9.23.3.28 x0data

```
double* x0data
```

state initialisation

Definition at line 116 of file udata.h.

### 9.23.3.29 `sx0data`

```
double* sx0data
```

sensitivity initialisation

Definition at line 119 of file `udata.h`.

### 9.23.3.30 `ordering`

```
int ordering
```

state ordering

Definition at line 122 of file `udata.h`.

## 10 File Documentation

### 10.1 `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.*

#### 10.1.1 Function Documentation

##### 10.1.1.1 `amiwrap()`

```
noret::substitute amiwrap (  
    matlabtypesubstitute varargin )
```

## Parameters

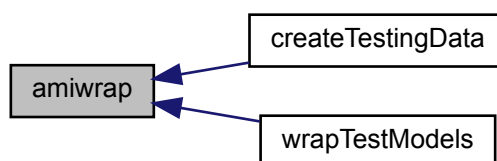
<i>varargin</i>	<pre>amiwrap ( modelname, symfun, tdir, o2flag )</pre> <p><i>Required Parameters for varargin:</i></p> <ul style="list-style-type: none"> <li>• modelname specifies the name of the model which will be later used for the naming of the simulation file</li> <li>• symfun specifies a function which executes model definition see <a href="#">Model Definition</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.

Here is the caller graph for this function:



## 10.2 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.*

## 10.2.1 Function Documentation

## 10.2.1.1 SBML2AMICI()

```
noret::substitute SBML2AMICI (
    matlabtypesubstitute filename,
    matlabtypesubstitute modelname )
```

## Parameters

<i>filename</i>	name of the SBML file (withouth 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:



### 10.3 SBMLImporter/computeBracketLevel.m File Reference

Compute the bracket level for the input string *cstr*. The bracket level is computed for every char in *cstr* and indicates how many brackets have been opened up to this point. The bracket level is useful to parse the arguments of functions in *cstr*. For this purpose functions will have the same bracket level as the opening bracket of the corresponding function call.

## Functions

- `mlhsInnerSubst<::int> computeBracketLevel (::*char cstr)`

*Compute the bracket level for the input string cstr. The bracket level is computed for every char in cstr and indicates how many brackets have been opened up to this point. The bracket level is useful to parse the arguments of functions in cstr. For this purpose functions will have the same bracket level as the opening bracket of the corresponding function call.*

#### 10.3.1 Function Documentation

##### 10.3.1.1 computeBracketLevel()

```
mlhsInnerSubst<::int> computeBracketLevel (
    ::*char cstr )
```

## Parameters

<i>cstr</i>	input string
-------------	--------------

## Return values

<i>brl</i>	bracket levels
------------	----------------

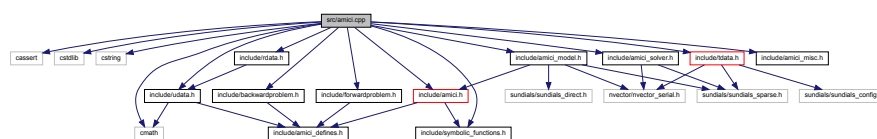
Definition at line 17 of file computeBracketLevel.m.

## 10.4 src/amici.cpp File Reference

core routines for integration

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

Include dependency graph for amici.cpp:



## Macros

- `#define _USE_MATH_DEFINES /* MS definition of PI and other constants */`
- `#define M_PI 3.14159265358979323846`

## Functions

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

## Variables

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

## 10.4.1 Function Documentation

### 10.4.1.1 runAmiciSimulation()

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

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

#### Parameters

in	<i>udata</i>	pointer to user data object <b>Type:</b> <a href="#">UserData</a>
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> <a href="#">ExpData</a>
in	<i>rdata</i>	pointer to return data object <b>Type:</b> <a href="#">ReturnData</a>
in	<i>model</i>	pointer to model specification object <b>Type:</b> <a href="#">Model</a>

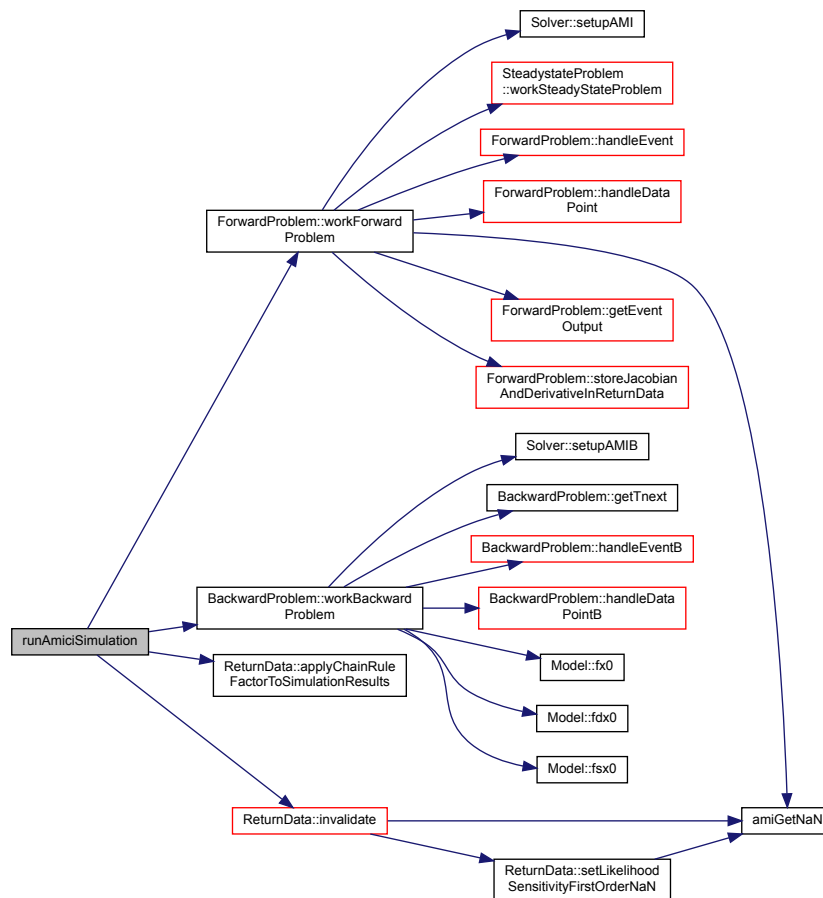
#### Returns

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

Definition at line 39 of file amici.cpp.



Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.4.1.2 printErrMsgIdAndTxt()

```

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

printErrMsgIdAndTxt prints a specified error message associated to the specified identifier

**Parameters**

in	<i>identifier</i>	error identifier <b>Type:</b> char
in	<i>msg</i>	error message <b>Type:</b> char

**Returns**

void

Definition at line 77 of file amici.cpp.

**10.4.1.3 printWarnMsgIdAndTxt()**

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

printErrMsgIdAndTxt prints a specified warning message associated to the specified identifier

**Parameters**

in	<i>identifier</i>	warning identifier <b>Type:</b> char
in	<i>msg</i>	warning message <b>Type:</b> char

**Returns**

void

Definition at line 88 of file amici.cpp.

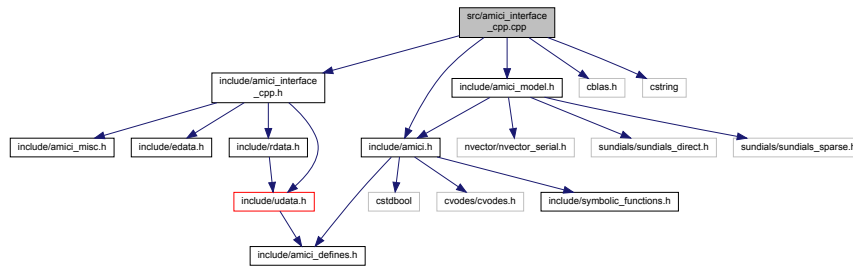
**10.5 src/amici\_interface\_cpp.cpp File Reference**

core routines for cpp interface

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

```
#include <cstring>
```

Include dependency graph for amici\_interface\_cpp.cpp:



## Functions

- [ReturnData](#) \* [getSimulationResults](#) ([Model](#) \*model, [UserData](#) \*udata, const [ExpData](#) \*edata)
- void [amici\\_dgemm](#) (AMICI\_BLAS\_LAYOUT layout, AMICI\_BLAS\_TRANSPOSE TransA, AMICI\_BLAS\_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double \*A, const int lda, const double \*B, const int ldb, const double beta, double \*C, const int ldc)
- void [amici\\_dgemv](#) (AMICI\_BLAS\_LAYOUT layout, AMICI\_BLAS\_TRANSPOSE TransA, const int M, const int N, const double alpha, const double \*A, const int lda, const double \*X, const int incX, const double beta, double \*Y, const int incY)

### 10.5.1 Function Documentation

#### 10.5.1.1 getSimulationResults()

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

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

#### Parameters

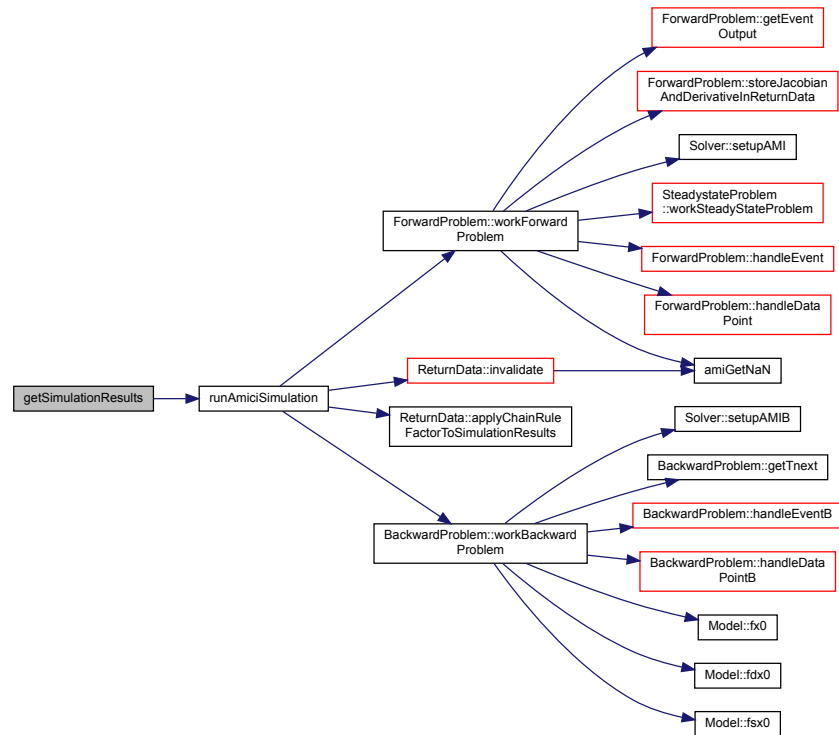
in	<i>model</i>	pointer to the model object, this is necessary to perform dimension checks <b>Type:</b> <a href="#">Model</a>
in	<i>udata</i>	pointer to user data object <b>Type:</b> <a href="#">UserData</a>
in	<i>edata</i>	pointer to experimental data object <b>Type:</b> <a href="#">ExpData</a>

#### Returns

rdata pointer to return data object  
**Type:** [ReturnData](#)

Definition at line 31 of file amici\_interface\_cpp.cpp.

Here is the call graph for this function:



### 10.5.1.2 amici\_dgemm()

```

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

```

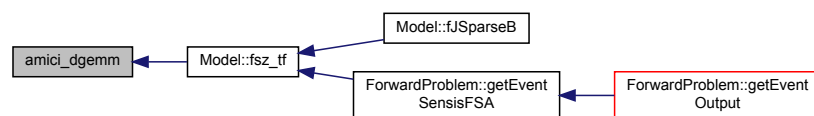
`amici_dgemm` provides an interface to the blas matrix matrix multiplication routine `dgemm`. This routines computes  $C = \alpha * A * B + \beta * C$  with  $A$ : [MxK]  $B$ : [KxN]  $C$ : [MxN]

## Parameters

in	<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)

Definition at line 60 of file amici\_interface\_cpp.cpp.

Here is the caller graph for this function:



## 10.5.1.3 amici\_dgemv()

```

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

```

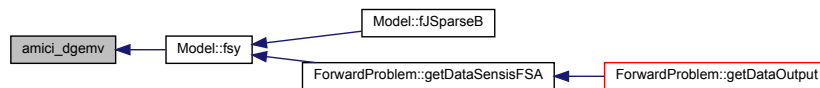
`amici_dgemm` provides an interface to the blas matrix vector multiplication routine `dgemv`. This routines computes  $y = \alpha * A * x + \beta * y$  with A: [MxK] B:[KxN] C:[MxN]

## Parameters

in	<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

Definition at line 83 of file amici\_interface\_cpp.cpp.

Here is the caller graph for this function:



## 10.6 src/amici\_interface\_matlab.cpp File Reference

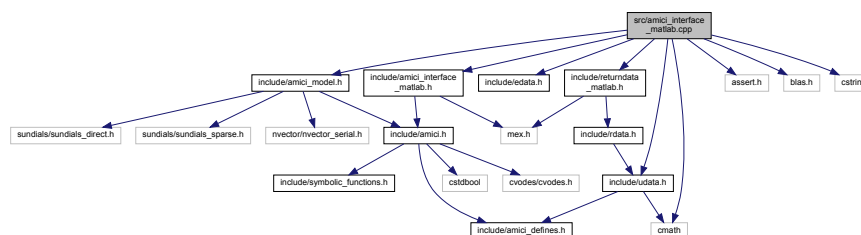
core routines for mex interface

```

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

```

Include dependency graph for amici\_interface\_matlab.cpp:



## Macros

- `#define _USE_MATH_DEFINES` /\* MS definition of PI and other constants \*/
- `#define M_PI` 3.14159265358979323846
- `#define readOptionScalar(OPTION, TYPE)`
- `#define readOptionData(OPTION)`

## Functions

- void `mexFunction` (int nlhs, mxArray \*plhs[], int nrhs, const mxArray \*prhs[])
- `UserData * userDataFromMatlabCall` (const mxArray \*prhs[], int nrhs, `Model` \*model)  
*userDataFromMatlabCall extracts information from the matlab call and returns the corresponding `UserData` struct*
- char `amici_blasCBlasTransToBlasTrans` (AMICI\_BLAS\_TRANSPOSE trans)
- void `amici_dgemm` (AMICI\_BLAS\_LAYOUT layout, AMICI\_BLAS\_TRANSPOSE TransA, AMICI\_BLAS\_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double \*A, const int lda, const double \*B, const int ldb, const double beta, double \*C, const int ldc)
- void `amici_dgemv` (AMICI\_BLAS\_LAYOUT layout, AMICI\_BLAS\_TRANSPOSE TransA, const int M, const int N, const double alpha, const double \*A, const int lda, const double \*X, const int incX, const double beta, double \*Y, const int incY)
- `ExpData * expDataFromMatlabCall` (const mxArray \*prhs[], const `UserData` \*udata, `Model` \*model)

### 10.6.1 Detailed Description

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

### 10.6.2 Macro Definition Documentation

#### 10.6.2.1 readOptionScalar

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

#### Value:

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

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

Definition at line 30 of file `amici_interface_matlab.cpp`.

### 10.6.2.2 readOptionData

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

**Value:**

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

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

Definition at line 43 of file amici\_interface\_matlab.cpp.

## 10.6.3 Function Documentation

### 10.6.3.1 mexFunction()

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

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

#### Parameters

in	<i>nlhs</i>	number of output arguments of the matlab call <b>Type:</b> int
out	<i>plhs</i>	pointer to the array of output arguments <b>Type:</b> mxArray
in	<i>nrhs</i>	number of input arguments of the matlab call <b>Type:</b> int
in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> mxArray

#### Returns

void

Definition at line 65 of file amici\_interface\_matlab.cpp.



## 10.6.3.2 userDataFromMatlabCall()

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

userDataFromMatlabCall parses the input from the matlab call and writes it to an [UserData](#) class object

## Parameters

in	<i>nrhs</i>	number of input arguments of the matlab call <b>Type:</b> int
in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> mxArray
in	<i>model</i>	pointer to the model object, this is necessary to perform dimension checks <b>Type:</b> <a href="#">Model</a>

## Returns

udata pointer to user data object  
**Type:** [UserData](#)

Definition at line 116 of file amici\_interface\_matlab.cpp.

## 10.6.3.3 amici\_blasCBlasTransToBlasTrans()

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

amici\_blasCBlasTransToBlasTrans translates AMICI\_BLAS\_TRANSPOSE values to CBlas readable strings

## Parameters

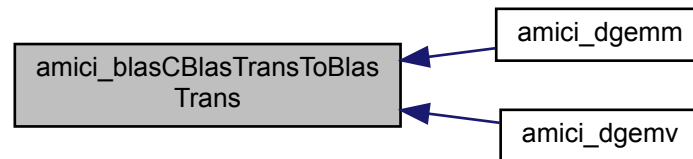
in	<i>trans</i>	flag indicating transposition and complex conjugation <b>Type:</b> AMICI_BLAS_TRANSPOSE
----	--------------	--

## Returns

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

Definition at line 323 of file amici\_interface\_matlab.cpp.

Here is the caller graph for this function:



#### 10.6.3.4 amici\_dgemm()

```

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

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

##### Parameters

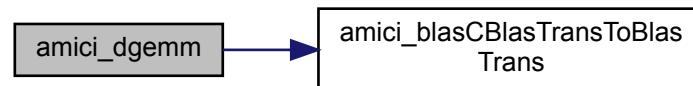
in	<i>layout</i>	always needs to be AMICI_BLAS_ColMajor.
in	<i>TransA</i>	flag indicating whether A should be transposed before multiplication
in	<i>TransB</i>	flag indicating whether B should be transposed before multiplication
in	<i>M</i>	number of rows in A/C
in	<i>N</i>	number of columns in B/C
in	<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)

**Returns**

void

Definition at line 355 of file amici\_interface\_matlab.cpp.

Here is the call graph for this function:

**10.6.3.5 amici\_dgemv()**

```

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

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

**Parameters**

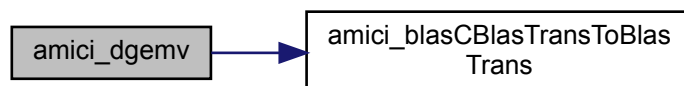
in	<i>layout</i>	always needs to be AMICI_BLAS_ColMajor.
in	<i>TransA</i>	flag indicating whether A should be transposed before multiplication
in	<i>M</i>	number of rows in A
in	<i>N</i>	number of columns in A
in	<i>alpha</i>	coefficient alpha
in	<i>A</i>	matrix A
in	<i>lda</i>	leading dimension of A (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

**Returns**

void

Definition at line 398 of file amici\_interface\_matlab.cpp.

Here is the call graph for this function:

**10.6.3.6 expDataFromMatlabCall()**

```

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

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

**Parameters**

in	<i>prhs</i>	pointer to the array of input arguments <b>Type:</b> mxArray
in	<i>udata</i>	pointer to user data object <b>Type:</b> <a href="#">UserData</a>
in	<i>model</i>	pointer to the model object, this is necessary to perform dimension checks <b>Type:</b> <a href="#">Model</a>

**Returns**

edata pointer to experimental data object  
**Type:** [ExpData](#)

Definition at line 426 of file amici\_interface\_matlab.cpp.

**10.7 src/spline.cpp File Reference**

definition of spline functions

## Functions

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

### 10.7.1 Detailed Description

#### Author

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

### 10.7.2 Function Documentation

#### 10.7.2.1 spline()

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

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

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

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

#### Parameters

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

## Return values

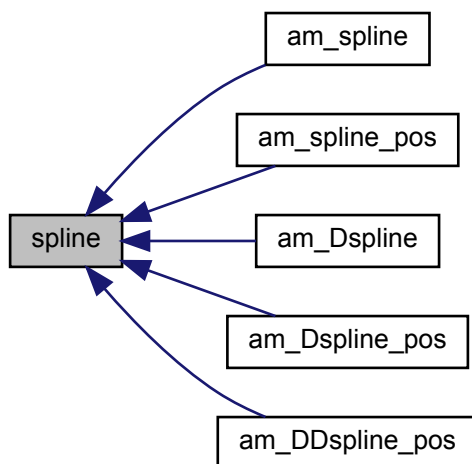
0	normal return
1	less than two data points; cannot interpolate
2	x[] are not in ascending order

## Notes

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

Here is the caller graph for this function:



## 10.7.2.2 seval()

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

Evaluate the cubic spline function

$S(x) = y[i] + b[i] * w + c[i] * w^2 + d[i] * w^3$  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

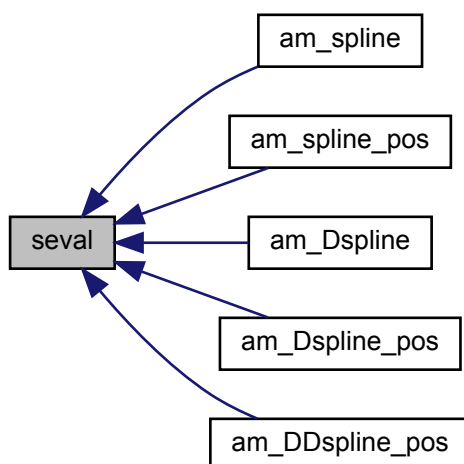
the value of the spline function at *u*

## 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 197 of file spline.cpp.

Here is the caller graph for this function:



### 10.7.2.3 sinteg()

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

Integrate the cubic spline function

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

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.

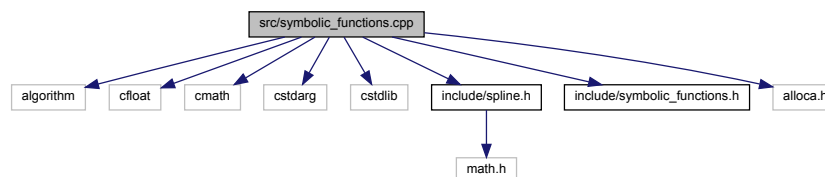
Definition at line 258 of file spline.cpp.

**10.8 src/symbolic\_functions.cpp File Reference**

definition of symbolic functions

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

Include dependency graph for symbolic\_functions.cpp:

**Functions**

- int [amilsNaN](#) (double what)
- int [amilsInf](#) (double what)
- double [amiGetNaN](#) ()
- double [amilog](#) (double x)
- double [dirac](#) (double x)
- double [heaviside](#) (double x)
- double [sign](#) (double x)
- double [am\\_min](#) (double a, double b, double c)
- double [Dam\\_min](#) (int id, double a, double b, double c)
- double [am\\_max](#) (double a, double b, double c)
- double [Dam\\_max](#) (int id, double a, double b, double c)
- double [am\\_spline](#) (double t, int num,...)
- double [am\\_spline\\_pos](#) (double t, int num,...)
- double [am\\_Dspline](#) (int id, double t, int num,...)
- double [am\\_Dspline\\_pos](#) (int id, double t, int num,...)
- double [am\\_DDspline](#) (int id1, int id2, double t, int num,...)
- double [am\\_DDspline\\_pos](#) (int id1, int id2, double t, int num,...)

### 10.8.1 Detailed Description

This file contains definitions of various symbolic functions which

### 10.8.2 Function Documentation

#### 10.8.2.1 amIsNaN()

```
int amIsNaN (
    double what )
```

c++ interface to the isNaN function

#### Parameters

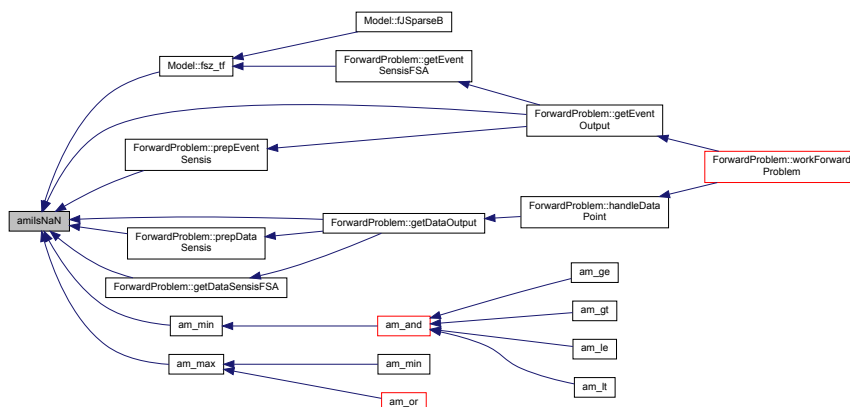
<i>what</i>	argument
-------------	----------

#### Returns

isnan(what)

Definition at line 32 of file symbolic\_functions.cpp.

Here is the caller graph for this function:



#### 10.8.2.2 amIsInf()

```
int amIsInf (
    double what )
```

c++ interface to the isinf function

**Parameters**

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

**Returns**

isnan(what)

Definition at line 43 of file symbolic\_functions.cpp.

**10.8.2.3 amiGetNaN()**

```
double amiGetNaN ( )
```

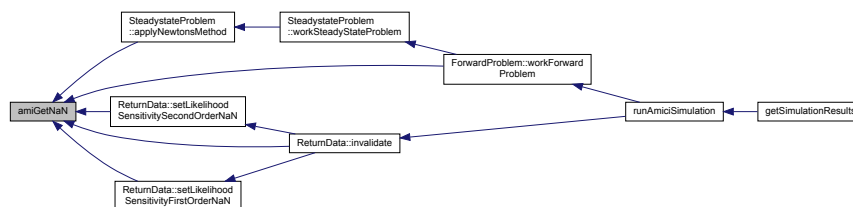
function returning nan

**Returns**

NaN

Definition at line 53 of file symbolic\_functions.cpp.

Here is the caller graph for this function:

**10.8.2.4 amilog()**

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

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

**Parameters**

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

**Returns**

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

Definition at line 65 of file symbolic\_functions.cpp.

**10.8.2.5 dirac()**

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

c implementation of matlab function dirac

**Parameters**

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

**Returns**

if( $x == 0$ ) then  $\text{INF}$  else 0

Definition at line 80 of file symbolic\_functions.cpp.

Here is the caller graph for this function:

**10.8.2.6 heaviside()**

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

c implementation of matlab function heaviside

**Parameters**

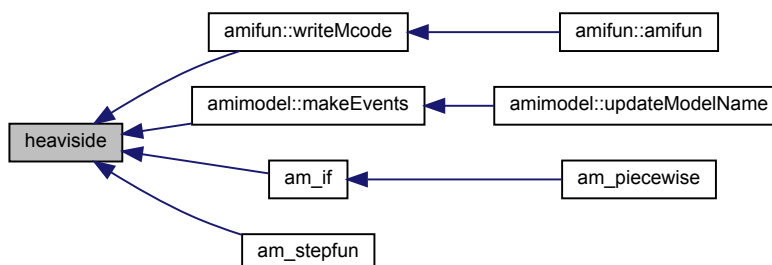
x	argument
---	----------

**Returns**

if( $x > 0$ ) then 1 else 0

Definition at line 95 of file symbolic\_functions.cpp.

Here is the caller graph for this function:



#### 10.8.2.7 sign()

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

c implementation of matlab function sign

##### Parameters

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

##### Returns

0

**Type:** double

Definition at line 110 of file symbolic\_functions.cpp.

#### 10.8.2.8 am\_min()

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

c implementation of matlab function min

##### Parameters

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

**Returns**

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

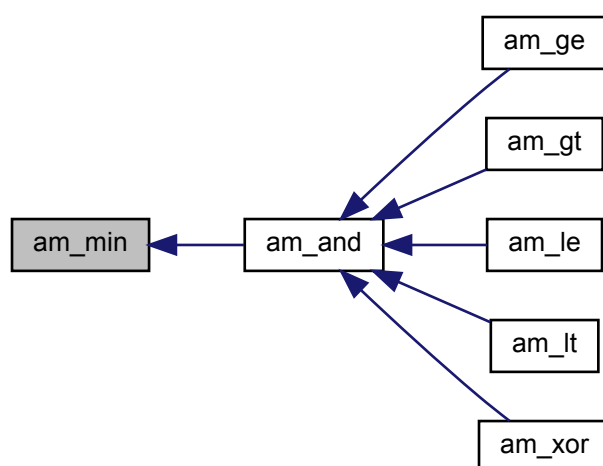
**Type:** double

Definition at line 131 of file symbolic\_functions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

**10.8.2.9 Dam\_min()**

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

parameter derivative of c implementation of matlab function min

## Parameters

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

## Returns

id == 1: if( $a < b$ ) then 1 else 0

**Type:** double

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

**Type:** double

Definition at line 154 of file symbolic\_functions.cpp.

## 10.8.2.10 am\_max()

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

c implementation of matlab function max

## Parameters

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

## Returns

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

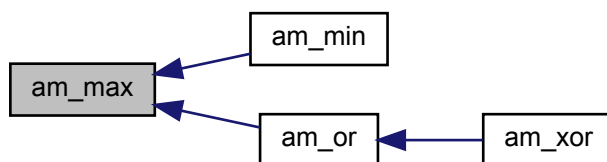
**Type:** double

Definition at line 179 of file symbolic\_functions.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 10.8.2.11 Dam\_max()

```

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

parameter derivative of c implementation of matlab function max

##### Parameters

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

##### Returns

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



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

Definition at line 202 of file symbolic\_functions.cpp.

#### 10.8.2.12 am\_spline()

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

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

##### Parameters

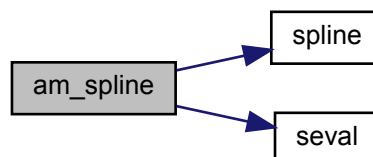
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

##### Returns

spline(t)

Definition at line 229 of file symbolic\_functions.cpp.

Here is the call graph for this function:



#### 10.8.2.13 am\_spline\_pos()

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

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

**Parameters**

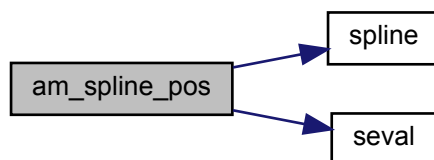
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

**Returns**

spline(t)

Definition at line 280 of file symbolic\_functions.cpp.

Here is the call graph for this function:

**10.8.2.14 am\_Dspline()**

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

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

**Parameters**

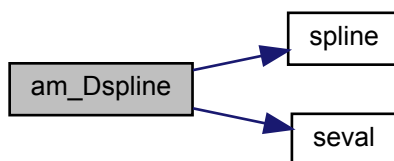
<i>id</i>	index of node to which the derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

**Returns**

dsplinedp(t)

Definition at line 332 of file symbolic\_functions.cpp.

Here is the call graph for this function:



#### 10.8.2.15 am\_Dspline\_pos()

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

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

##### Parameters

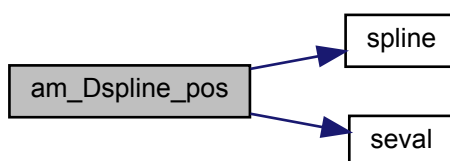
<i>id</i>	index of node to which the derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number of spline nodes

##### Returns

`dsplinedp(t)`

Definition at line 388 of file `symbolic_functions.cpp`.

Here is the call graph for this function:



### 10.8.2.16 am\_DD spline()

```
double am_DD spline (
    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

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

#### Returns

ddspline(t)

Definition at line 453 of file symbolic\_functions.cpp.

### 10.8.2.17 am\_DD spline\_pos()

```
double am_DD spline_pos (
    int id1,
    int id2,
    double t,
    int num,
    ... )
```

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

#### Parameters

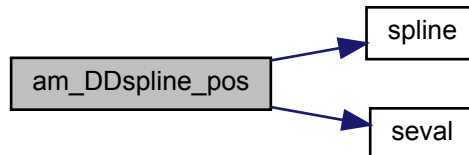
<i>id1</i>	index of node to which the first derivative of the corresponding spline coefficient should be computed
<i>id2</i>	index of node to which the second derivative of the corresponding spline coefficient should be computed
<i>t</i>	point at which the spline should be evaluated
<i>num</i>	number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

**Returns**

ddspline(t)

Definition at line 468 of file symbolic\_functions.cpp.

Here is the call graph for this function:

**10.9 symbolic/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*

**10.9.1 Function Documentation****10.9.1.1 am\_and()**

```
mlhsInnerSubst< matlabtypesubstitute > am_and (
    ::sym a,
    ::sym b )
```

**Parameters**

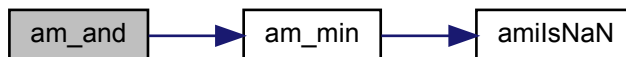
<i>a</i>	first input parameter
<i>b</i>	second input parameter

**Return values**

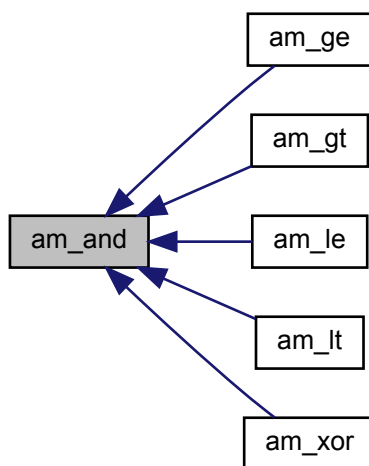
<i>fun</i>	logical value, negative for false, positive for true
------------	--

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:



## 10.10 symbolic/am\_eq.m File Reference

`am_eq` is currently a placeholder that simply produces an error message

### Functions

- `mlhsInnerSubst< matlabtypesubstitute > am_eq (matlabtypesubstitute varargin)`  
*am\_eq is currently a placeholder that simply produces an error message*

### 10.10.1 Function Documentation

#### 10.10.1.1 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.

## 10.11 symbolic/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  $\text{and}(\text{varargin}\{1\} \geq \text{varargin}\{2\}, \text{varargin}\{2\} \geq \text{varargin}\{3\}, \dots)$

## 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  $\text{and}(\text{varargin}\{1\} \geq \text{varargin}\{2\}, \text{varargin}\{2\} \geq \text{varargin}\{3\}, \dots)$*

## 10.11.1 Function Documentation

## 10.11.1.1 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:



## 10.12 symbolic/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  $\text{and}(\text{varargin}\{1\} > \text{varargin}\{2\}, \text{varargin}\{2\} > \text{varargin}\{3\}, \dots)$

### 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  $\text{and}(\text{varargin}\{1\} > \text{varargin}\{2\}, \text{varargin}\{2\} > \text{varargin}\{3\}, \dots)$*

### 10.12.1 Function Documentation

#### 10.12.1.1 am\_gt()

```
mlhsInnerSubst< matlabtypesubstitute > am_gt (
    ::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_gt.m`.



Here is the call graph for this function:



## 10.13 symbolic/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*

### 10.13.1 Function Documentation

#### 10.13.1.1 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

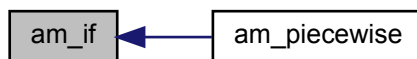
<i>fun</i>	if condition is true truepart, else falsepart
------------	---

Definition at line 17 of file am\_if.m.

Here is the call graph for this function:



Here is the caller graph for this function:



## 10.14 symbolic/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}, ...)`*

### 10.14.1 Function Documentation

#### 10.14.1.1 `am_le()`

```
mlhsInnerSubst< matlabtypesubstitute > am_le (
    ::sym varargin )
```

#### Parameters

<code>varargin</code>	chain of input parameters
-----------------------	---------------------------

## Return values

<i>fun</i>	$a \leq b$ 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:



## 10.15 symbolic/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  $\text{and}(\text{varargin}\{1\} < \text{varargin}\{2\}, \text{varargin}\{2\} < \text{varargin}\{3\}, \dots)$

## 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  $\text{and}(\text{varargin}\{1\} < \text{varargin}\{2\}, \text{varargin}\{2\} < \text{varargin}\{3\}, \dots)$*

## 10.15.1 Function Documentation

## 10.15.1.1 am\_lt()

```
mlhsInnerSubst < matlabtypesubstitute > am_lt (
    ::sym varargin )
```

## Parameters

<i>varargin</i>	chain of input parameters
-----------------	---------------------------

## Return values

<i>fun</i>	$a < b$ logical value, negative for false, positive for true
------------	--

Definition at line 17 of file am\_lt.m.

Here is the call graph for this function:



## 10.16 symbolic/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*

### 10.16.1 Function Documentation

#### 10.16.1.1 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`.

## 10.17 symbolic/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*

### 10.17.1 Function Documentation

#### 10.17.1.1 am\_min()

```
mlhsInnerSubst< matlabtypesubstitute > am_min (
    ::sym a,
    ::sym b )
```

#### Parameters

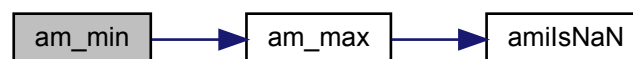
<i>a</i>	first input parameter
<i>b</i>	second input parameter

#### Return values

<i>fun</i>	minimum of a and b
------------	--------------------

Definition at line 17 of file `am_min.m`.

Here is the call graph for this function:



## 10.18 symbolic/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*

## 10.18.1 Function Documentation

### 10.18.1.1 am\_or()

```
mlhsInnerSubst< matlabtypesubstitute > am_or (
    ::sym a,
    ::sym b )
```

#### Parameters

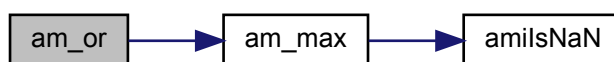
<i>a</i>	first input parameter
<i>b</i>	second input parameter

#### 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:



## 10.19 symbolic/am\_piecewise.m File Reference

am\_piecewise is the amici implementation of the mathml piecewise function

## Functions

- `mlhsInnerSubst< matlabtypesubstitute > am_pieewise` (`matlabtypesubstitute piece`, `matlabtypesubstitute condition`, `matlabtypesubstitute default`)

*am\_pieewise is the amici implementation of the mathml piecewise function*

### 10.19.1 Function Documentation

#### 10.19.1.1 am\_pieewise()

```
mlhsInnerSubst< matlabtypesubstitute > am_pieewise (
    matlabtypesubstitute piece,
    matlabtypesubstitute condition,
    matlabtypesubstitute default )
```

#### Parameters

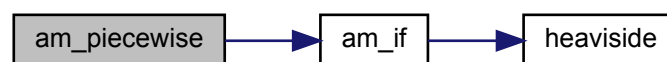
<i>piece</i>	value if condition is true
<i>condition</i>	logical value
<i>default</i>	value if condition is false

#### Return values

<i>fun</i>	return value, piece if condition is true, default if not
------------	--

Definition at line 17 of file `am_pieewise.m`.

Here is the call graph for this function:



## 10.20 symbolic/am\_stepfun.m File Reference

`am_stepfun` is the amici implementation of the step function

## Functions

- `mlhsInnerSubst< matlabtypesubstitute > am_stepfun` (`::sym t`, `matlabtypesubstitute tstart`, `matlabtypesubstitute vstart`, `matlabtypesubstitute tend`, `matlabtypesubstitute vend`)

*am\_stepfun is the amici implementation of the step function*

## 10.20.1 Function Documentation

### 10.20.1.1 am\_stepfun()

```
mlhsInnerSubst< matlabtypesubstitute > am_stepfun (
    ::sym t,
    matlabtypesubstitute tstart,
    matlabtypesubstitute vstart,
    matlabtypesubstitute tend,
    matlabtypesubstitute vend )
```

#### Parameters

<i>t</i>	input variable
<i>tstart</i>	input variable value at which the step starts
<i>vstart</i>	value during the step
<i>tend</i>	input variable value at which the step end
<i>vend</i>	value after the step

#### Return values

<i>fun</i>	0 before tstart, vstart between tstart and tend and vend after tend
------------	---

Definition at line 17 of file am\_stepfun.m.

Here is the call graph for this function:



## 10.21 symbolic/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*



## 10.21.1 Function Documentation

## 10.21.1.1 am\_xor()

```
mlhsInnerSubst< matlabtypesubstitute > am_xor (
    ::sym a,
    ::sym b )
```

## Parameters

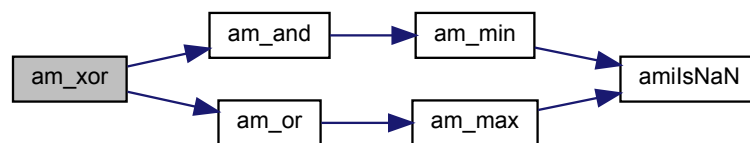
<i>a</i>	first input parameter
<i>b</i>	second input parameter

## 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:



## 10.22 tests/cpptest/createTestingData.m File Reference

createTestingData runs simulation on all test models and writes results as hdf5. currently necessary for continuous integration to yield meaningful results

## Functions

- noret::substitute [createTestingData](#) ()  
*createTestingData runs simulation on all test models and writes results as hdf5. currently necessary for continuous integration to yield meaningful results*

## 10.23 tests/cpptest/wrapTestModels.m File Reference

wrapTestModels calls amiwrap on all test models. currently necessary for continuous integrations to yield meaningful results

## Functions

- `noret::substitute wrapTestModels ()`

*wrapTestModels calls amiwrap on all test models. currently necessary for continuous integrations to yield meaningful results*

## Index

~ReturnData

ReturnData, [158](#)

adjoint

amimodel, [47](#)

am\_DDspline

symbolic\_functions.cpp, [258](#)

am\_DDspline\_pos

symbolic\_functions.cpp, [258](#)

am\_Dspline

symbolic\_functions.cpp, [256](#)

am\_Dspline\_pos

symbolic\_functions.cpp, [257](#)

am\_and

am\_and.m, [259](#)

am\_and.m

am\_and, [259](#)

am\_eq

am\_eq.m, [260](#)

am\_eq.m

am\_eq, [260](#)

am\_ge

am\_ge.m, [261](#)

am\_ge.m

am\_ge, [261](#)

am\_gt

am\_gt.m, [262](#)

am\_gt.m

am\_gt, [262](#)

am\_if

am\_if.m, [263](#)

am\_if.m

am\_if, [263](#)

am\_le

am\_le.m, [264](#)

am\_le.m

am\_le, [264](#)

am\_lt

am\_lt.m, [265](#)

am\_lt.m

am\_lt, [265](#)

am\_max

am\_max.m, [266](#)

symbolic\_functions.cpp, [253](#)

am\_max.m

am\_max, [266](#)

am\_min

am\_min.m, [267](#)

symbolic\_functions.cpp, [251](#)

am\_min.m

am\_min, [267](#)

am\_or

am\_or.m, [268](#)

am\_or.m

am\_or, [268](#)

am\_piecewise

am\_piecewise.m, [269](#)

am\_piecewise.m

am\_piecewise, [269](#)

am\_spline

symbolic\_functions.cpp, [255](#)

am\_spline\_pos

symbolic\_functions.cpp, [255](#)

am\_stepfun

am\_stepfun.m, [270](#)

am\_stepfun.m

am\_stepfun, [270](#)

am\_xor

am\_xor.m, [271](#)

am\_xor.m

am\_xor, [271](#)

ami\_mem

Solver, [194](#)

amiGetNaN

symbolic\_functions.cpp, [249](#)

amilsInf

symbolic\_functions.cpp, [248](#)

amilsNaN

symbolic\_functions.cpp, [248](#)

amici.cpp

printErrMsgIdAndTxt, [231](#)

printWarnMsgIdAndTxt, [232](#)

runAmiciSimulation, [230](#)

amici\_blasCBlasTransToBlasTrans

amici\_interface\_matlab.cpp, [239](#)

amici\_dgemm

amici\_interface\_cpp.cpp, [234](#)

amici\_interface\_matlab.cpp, [240](#)

amici\_dgemv

amici\_interface\_cpp.cpp, [235](#)

amici\_interface\_matlab.cpp, [241](#)

amici\_interface\_cpp.cpp

amici\_dgemm, [234](#)

amici\_dgemv, [235](#)

getSimulationResults, [233](#)

amici\_interface\_matlab.cpp

amici\_blasCBlasTransToBlasTrans, [239](#)

amici\_dgemm, [240](#)

amici\_dgemv, [241](#)

expDataFromMatlabCall, [242](#)

mexFunction, [238](#)

readOptionData, [237](#)

readOptionScalar, [237](#)

userDataFromMatlabCall, [238](#)

amidata, [15](#)

amidata, [17](#)

condition, [20](#)

ne, [18](#)

nk, [18](#)

nt, [17](#)

- ny, [17](#)
- nz, [18](#)
- Sigma\_Y, [19](#)
- Sigma\_Z, [19](#)
- t, [18](#)
- Y, [19](#)
- Z, [19](#)
- amievent, [20](#)
  - amievent, [21](#)
  - bolus, [22](#)
  - hflag, [23](#)
  - setHflag, [21](#)
  - trigger, [22](#)
  - z, [22](#)
- amifun, [23](#)
  - amifun, [24](#)
  - argstr, [33](#)
  - cvar, [33](#)
  - deps, [33](#)
  - fargstr, [33](#)
  - funstr, [33](#)
  - gccode, [28](#)
  - getArgs, [29](#)
  - getCVar, [31](#)
  - getDeps, [29](#)
  - getFArgs, [30](#)
  - getNVecs, [30](#)
  - getSensiFlag, [31](#)
  - nvecs, [34](#)
  - printLocalVars, [25](#)
  - sensiflag, [34](#)
  - strsym, [32](#)
  - strsym\_old, [32](#)
  - sym, [32](#)
  - sym\_noopt, [32](#)
  - writeCcode, [26](#)
  - writeCcode\_sensi, [26](#)
  - writeMcode, [27](#)
- amilog
  - symbolic\_functions.cpp, [249](#)
- amimodel, [34](#)
  - adjoint, [47](#)
  - amimodel, [38](#)
  - augmento2, [44](#)
  - augmento2vec, [44](#)
  - cfun, [55](#)
  - checkDeps, [42](#)
  - colptrs, [53](#)
  - colptrsB, [53](#)
  - compileAndLinkModel, [45](#)
  - compileC, [40](#)
  - coptim, [54](#)
  - debug, [47](#)
  - event, [46](#)
  - forward, [48](#)
  - fun, [46](#)
  - funs, [54](#)
  - generateC, [39](#)
  - generateM, [41](#)
  - getFun, [41](#)
  - HTable, [47](#)
  - id, [51](#)
  - lbw, [52](#)
  - loadOldHashes, [43](#)
  - maxflag, [56](#)
  - mfuns, [54](#)
  - minflag, [56](#)
  - modelName, [47](#)
  - ndwdp, [57](#)
  - ndwdx, [57](#)
  - nevent, [50](#)
  - ng, [50](#)
  - nk, [50](#)
  - nnz, [52](#)
  - np, [50](#)
  - nw, [56](#)
  - nx, [49](#)
  - nxtrue, [49](#)
  - ny, [49](#)
  - nytrue, [49](#)
  - nz, [51](#)
  - nztrue, [51](#)
  - o2flag, [55](#)
  - param, [54](#)
  - recompile, [55](#)
  - rowvals, [52](#)
  - rowvalsB, [53](#)
  - sparseidx, [52](#)
  - sparseidxB, [53](#)
  - splineflag, [56](#)
  - steadystate, [48](#)
  - sym, [46](#)
  - t0, [48](#)
  - ubw, [51](#)
  - updateModelName, [39](#)
  - updateRHS, [38](#)
  - wrap\_path, [55](#)
  - wtype, [48](#)
  - z2event, [56](#)
- amioption, [57](#)
  - amioption, [59](#)
  - atol, [60](#)
  - id, [66](#)
  - interpType, [62](#)
  - ism, [62](#)
  - iter, [61](#)
  - iterB, [62](#)
  - linsol, [61](#)
  - Imm, [61](#)
  - ImmB, [62](#)
  - maxsteps, [60](#)
  - newton\_maxlinsteps, [65](#)
  - newton\_maxsteps, [64](#)
  - newton\_precon, [64](#)
  - newton\_preeq, [65](#)
  - nmaxevent, [63](#)

- ordering, [63](#)
- pbar, [61](#)
- pscale, [65](#)
- qpositivex, [60](#)
- rtol, [60](#)
- sens\_ind, [60](#)
- sensi, [63](#)
- sensi\_meth, [63](#)
- ss, [64](#)
- stldet, [62](#)
- sx0, [64](#)
- tstart, [61](#)
- x0, [64](#)
- z2event, [65](#)
- amised, [66](#)
  - amised, [67](#)
  - datasym, [69](#)
  - model, [68](#)
  - modelName, [68](#)
  - outputcount, [69](#)
  - sedml, [68](#)
  - varidx, [69](#)
  - varsym, [69](#)
- amiwrap
  - amiwrap.m, [226](#)
- amiwrap.m, [226](#)
  - amiwrap, [226](#)
- applyChainRuleFactorToSimulationResults
  - ReturnData, [161](#)
- applyEventBolus
  - ForwardProblem, [92](#)
- applyEventSensiBolusFSA
  - ForwardProblem, [94](#)
- applyNewtonsMethod
  - SteadystateProblem, [196](#)
- argstr
  - amifun, [33](#)
- atol
  - amioption, [60](#)
  - UserData, [222](#)
- augmento2
  - amimodel, [44](#)
- augmento2vec
  - amimodel, [44](#)
- BackwardProblem, [70](#)
  - getTnext, [74](#)
  - handleDataPointB, [72](#)
  - handleEventB, [71](#)
  - updateHeavisideB, [73](#)
  - workBackwardProblem, [70](#)
- bolus
  - amievent, [22](#)
  - SBMLode, [188](#)
- cfun
  - amimodel, [55](#)
- checkDeps
  - amimodel, [42](#)
- checkODE
  - SBMLode, [184](#)
- chi2
  - ReturnData, [172](#)
- colptrs
  - amimodel, [53](#)
- colptrsB
  - amimodel, [53](#)
- compartment
  - SBMLode, [186](#)
- compileAndLinkModel
  - amimodel, [45](#)
- compileC
  - amimodel, [40](#)
- computeBracketLevel
  - computeBracketLevel.m, [228](#)
- computeBracketLevel.m
  - computeBracketLevel, [228](#)
- condition
  - amidata, [20](#)
  - SBMLode, [187](#)
- constant
  - SBMLode, [186](#)
- coptim
  - amimodel, [54](#)
- copyFromUserData
  - ReturnData, [161](#)
- cvar
  - amifun, [33](#)
- dJrzdsigma
  - TempData, [208](#)
- dJrzdz
  - TempData, [208](#)
- dJydp
  - TempData, [206](#)
- dJydsigma
  - TempData, [207](#)
- dJydx
  - TempData, [207](#)
- dJydy
  - TempData, [207](#)
- dJzdp
  - TempData, [207](#)
- dJzdsigma
  - TempData, [208](#)
- dJzdx
  - TempData, [208](#)
- dJzdz
  - TempData, [208](#)
- Dam\_max
  - symbolic\_functions.cpp, [254](#)
- Dam\_min
  - symbolic\_functions.cpp, [252](#)
- datasym
  - amised, [69](#)
- debug
  - amimodel, [47](#)
- deltaqB

- TempData, 213
- deltasx
  - TempData, 212
- deltax
  - TempData, 212
- deltaxB
  - TempData, 212
- deps
  - amifun, 33
- dfdx
  - TempData, 215
- dirac
  - symbolic\_functions.cpp, 250
- discs
  - TempData, 213
- drzdp
  - TempData, 209
- drzdx
  - TempData, 209
- dsigmaydp
  - TempData, 210
- dsigmazdp
  - TempData, 211
- dwdp
  - TempData, 214
- dwdx
  - TempData, 214
- dx
  - TempData, 204
- dx\_old
  - TempData, 204
- dxB
  - TempData, 205
- dxdotdp
  - ReturnData, 166
  - TempData, 214
- dydp
  - ReturnData, 166
  - TempData, 209
- dydx
  - ReturnData, 166
  - TempData, 210
- dzdp
  - TempData, 209
- dzdx
  - TempData, 209
- event
  - amimodel, 46
- ExpData, 75
  - ExpData, 75
  - mrz, 77
  - my, 76
  - mz, 77
  - setDefault, 76
  - sigmay, 77
  - sigmaz, 77
- expDataFromMatlabCall
  - amici\_interface\_matlab.cpp, 242
- fJBand
  - Model, 134
- fJBandB
  - Model, 135
- fJDiag
  - Model, 104
- fJSparse
  - Model, 133
- fJSparseB
  - Model, 137
- fJB
  - Model, 103
- fJrz
  - Model, 126
- fJv
  - Model, 104
- fJvB
  - Model, 136
- fJy
  - Model, 125
- fJz
  - Model, 126
- fargstr
  - amifun, 33
- fdJrzdsigma
  - Model, 131
- fdJrzdz
  - Model, 130
- fdJydsigma
  - Model, 128
- fdJydy
  - Model, 127
- fdJzdsigma
  - Model, 130
- fdJzdz
  - Model, 129
- fdeltaqB
  - Model, 121
- fdeltasx
  - Model, 119
- fdeltax
  - Model, 118
- fdeltaxB
  - Model, 120
- fdrzdp
  - Model, 114
- fdrzdx
  - Model, 114
- fdsigma\_ydp
  - Model, 123
- fdsigma\_zdp
  - Model, 124
- fdx0
  - Model, 100
- fdxdotdp
  - Model, 117
- fdydp
  - Model, 109

- fdydx
  - Model, [110](#)
- fdzdp
  - Model, [112](#)
- fdzdx
  - Model, [113](#)
- fJ
  - Model, [102](#)
- flux
  - SBMLode, [187](#)
- forward
  - amimodel, [48](#)
- ForwardProblem, [77](#)
  - applyEventBolus, [92](#)
  - applyEventSensiBolusFSA, [94](#)
  - getDataOutput, [88](#)
  - getDataSensisFSA, [91](#)
  - getEventOutput, [82](#)
  - getEventSensisFSA, [85](#)
  - handleDataPoint, [86](#)
  - handleEvent, [79](#)
  - prepDataSensis, [89](#)
  - prepEventSensis, [83](#)
  - storeJacobianAndDerivativeInReturnData, [81](#)
  - updateHeaviside, [95](#)
  - workForwardProblem, [78](#)
- fqBdot
  - Model, [117](#)
- freeFieldsOnDestruction
  - ReturnData, [173](#)
- froot
  - Model, [105](#)
- frz
  - Model, [106](#)
- fsdx0
  - Model, [101](#)
- fsigma\_y
  - Model, [122](#)
- fsigma\_z
  - Model, [123](#)
- fsrz
  - Model, [107](#)
- fstau
  - Model, [108](#)
- fsx0
  - Model, [100](#)
- fsxdot
  - Model, [132](#)
- fsy
  - Model, [138](#)
- fsz
  - Model, [112](#)
- fsz\_tf
  - Model, [139](#)
- fun
  - amimodel, [46](#)
- funmath
  - SBMLode, [188](#)
- funcs
  - amimodel, [54](#)
- funstr
  - amifun, [33](#)
- fx0
  - Model, [99](#)
- fxBdot
  - Model, [116](#)
- fxdot
  - Model, [115](#)
- fy
  - Model, [108](#)
- fz
  - Model, [111](#)
- gccode
  - amifun, [28](#)
- generateC
  - amimodel, [39](#)
- generateM
  - amimodel, [41](#)
- getArgs
  - amifun, [29](#)
- getCVar
  - amifun, [31](#)
- getDataOutput
  - ForwardProblem, [88](#)
- getDataSensisFSA
  - ForwardProblem, [91](#)
- getDeps
  - amifun, [29](#)
- getDiagnosis
  - Solver, [193](#)
- getDiagnosisB
  - Solver, [193](#)
- getEventOutput
  - ForwardProblem, [82](#)
- getEventSensisFSA
  - ForwardProblem, [85](#)
- getFArgs
  - amifun, [30](#)
- getFun
  - amimodel, [41](#)
- getNVecs
  - amifun, [30](#)
- getNewtonOutput
  - SteadystateProblem, [197](#)
- getNewtonSimulation
  - SteadystateProblem, [198](#)
- getSensiFlag
  - amifun, [31](#)
- getSimulationResults
  - amici\_interface\_cpp.cpp, [233](#)
- getSolver
  - Model, [98](#)
  - NewtonSolver, [147](#)
- getStep
  - NewtonSolver, [147](#)
  - NewtonSolverDense, [149](#)

- NewtonSolverIterative, [151](#)
  - NewtonSolverSparse, [153](#)
- getTnext
  - BackwardProblem, [74](#)
- getoptimized
  - optsym, [155](#)
- h
  - TempData, [212](#)
- h\_adata
  - TempData, [212](#)
- HTable
  - amimodel, [47](#)
- handleDataPoint
  - ForwardProblem, [86](#)
- handleDataPointB
  - BackwardProblem, [72](#)
- handleEvent
  - ForwardProblem, [79](#)
- handleEventB
  - BackwardProblem, [71](#)
- heaviside
  - symbolic\_functions.cpp, [250](#)
- hflag
  - amievent, [23](#)
- id
  - amimodel, [51](#)
  - amioption, [66](#)
- idlist
  - Model, [145](#)
- init
  - UserData, [219](#)
- initField1
  - ReturnData, [163](#)
  - ReturnDataMatlab, [179](#)
- initField2
  - ReturnData, [163](#)
  - ReturnDataMatlab, [180](#)
- initField3
  - ReturnData, [164](#)
  - ReturnDataMatlab, [180](#)
- initField4
  - ReturnData, [165](#)
  - ReturnDataMatlab, [181](#)
- initFields
  - ReturnData, [162](#)
  - ReturnDataMatlab, [179](#)
- initHeaviside
  - Model, [140](#)
- initState
  - SBMLode, [187](#)
- interpType
  - amioption, [62](#)
  - UserData, [224](#)
- invalidate
  - ReturnData, [159](#)
- irdiscs
  - TempData, [213](#)
- iroot
  - TempData, [215](#)
- ism
  - amioption, [62](#)
  - UserData, [224](#)
- iter
  - amioption, [61](#)
  - UserData, [225](#)
- iterB
  - amioption, [62](#)
- J
  - ReturnData, [166](#)
  - TempData, [213](#)
- Jtmp
  - TempData, [206](#)
- Jy
  - TempData, [206](#)
- Jz
  - TempData, [207](#)
- k
  - UserData, [221](#)
- knom
  - SBMLode, [189](#)
- kvolume
  - SBMLode, [187](#)
- lbw
  - amimodel, [52](#)
  - Model, [144](#)
- linsol
  - amioption, [61](#)
  - UserData, [224](#)
- linsolveSPBCG
  - SteadystateProblem, [198](#)
- llh
  - ReturnData, [172](#)
- llhS0
  - TempData, [206](#)
- Imm
  - amioption, [61](#)
  - UserData, [225](#)
- ImmB
  - amioption, [62](#)
- loadOldHashes
  - amimodel, [43](#)
- M
  - TempData, [214](#)
- maxflag
  - amimodel, [56](#)
- maxsteps
  - amioption, [60](#)
  - UserData, [223](#)
- mexFunction
  - amici\_interface\_matlab.cpp, [238](#)
- mfuns
  - amimodel, [54](#)



- minflag
  - amimodel, 56
- Model, 95
  - fJBand, 134
  - fJBandB, 135
  - fJDiag, 104
  - fJSparse, 133
  - fJSparseB, 137
  - fJB, 103
  - fJrz, 126
  - fJv, 104
  - fJvB, 136
  - fJy, 125
  - fJz, 126
  - fdJrzdsigma, 131
  - fdJrzdz, 130
  - fdJydsigma, 128
  - fdJydy, 127
  - fdJzdsigma, 130
  - fdJzdz, 129
  - fdeltaqB, 121
  - fdeltasx, 119
  - fdeltax, 118
  - fdeltaxB, 120
  - fdrzdp, 114
  - fdrzdx, 114
  - fdsigma\_ydp, 123
  - fdsigma\_zdp, 124
  - fdx0, 100
  - fdxdotdp, 117
  - fdydp, 109
  - fdydx, 110
  - fdzdp, 112
  - fdzdx, 113
  - fJ, 102
  - fqBdot, 117
  - froot, 105
  - frz, 106
  - fsdx0, 101
  - fsigma\_y, 122
  - fsigma\_z, 123
  - fsrz, 107
  - fstau, 108
  - fsx0, 100
  - fsxdot, 132
  - fsy, 138
  - fsz, 112
  - fsz\_tf, 139
  - fx0, 99
  - fxBdot, 116
  - fxdot, 115
  - fy, 108
  - fz, 111
  - getSolver, 98
  - idlist, 145
  - initHeaviside, 140
  - lbw, 144
  - Model, 97
  - ndwdp, 143
  - ndwdx, 143
  - ne, 143
  - nJ, 144
  - nk, 141
  - nnz, 144
  - np, 141
  - nw, 143
  - nx, 142
  - nxtrue, 142
  - ny, 142
  - nytrue, 142
  - nz, 142
  - nztrue, 143
  - o2mode, 144
  - ubw, 144
  - z2event, 145
- model
  - amised, 68
  - TempData, 217
- modelTest, 145
- modelName
  - amimodel, 47
  - amised, 68
- mrz
  - ExpData, 77
- mxsol
  - ReturnDataMatlab, 181
- my
  - ExpData, 76
- mz
  - ExpData, 77
- nan\_JDiag
  - TempData, 216
- nan\_JSparse
  - TempData, 216
- nan\_dxdotdp
  - TempData, 215
- nan\_J
  - TempData, 216
- nan\_qBdot
  - TempData, 217
- nan\_xBdot
  - TempData, 216
- nan\_xdot
  - TempData, 216
- ndwdp
  - amimodel, 57
  - Model, 143
- ndwdx
  - amimodel, 57
  - Model, 143
- ne
  - amidata, 18
  - Model, 143
  - ReturnData, 175
- nevent
  - amimodel, 50

- newton\_maxlinsteps
  - amioption, 65
  - UserData, 223
- newton\_maxsteps
  - amioption, 64
  - ReturnData, 176
  - UserData, 223
- newton\_numlinsteps
  - ReturnData, 172
- newton\_numsteps
  - ReturnData, 171
- newton\_precon
  - amioption, 64
  - UserData, 224
- newton\_preeq
  - amioption, 65
  - UserData, 223
- newton\_status
  - ReturnData, 171
- newton\_time
  - ReturnData, 171
- NewtonSolver, 146
  - getSolver, 147
  - getStep, 147
- NewtonSolverDense, 148
  - getStep, 149
- NewtonSolverIterative, 150
  - getStep, 151
- NewtonSolverSparse, 152
  - getStep, 153
- ng
  - amimodel, 50
- nJ
  - Model, 144
  - ReturnData, 175
- nk
  - amidata, 18
  - amimodel, 50
  - Model, 141
  - ReturnData, 173
- nmaxevent
  - amioption, 63
  - ReturnData, 175
  - UserData, 220
- nnz
  - amimodel, 52
  - Model, 144
- np
  - amimodel, 50
  - Model, 141
  - ReturnData, 173
- nplist
  - ReturnData, 175
  - TempData, 215
  - UserData, 220
- nroots
  - TempData, 211
- nt
  - amidata, 17
  - ReturnData, 176
  - UserData, 221
- numerrtestfails
  - ReturnData, 170
- numerrtestfailsB
  - ReturnData, 170
- numnonlinsolvconvfails
  - ReturnData, 170
- numnonlinsolvconvfailsB
  - ReturnData, 171
- numrhsevals
  - ReturnData, 170
- numrhsevalsB
  - ReturnData, 170
- numsteps
  - ReturnData, 169
- numstepsB
  - ReturnData, 169
- nvecs
  - amifun, 34
- nw
  - amimodel, 56
  - Model, 143
- nx
  - amimodel, 49
  - Model, 142
  - ReturnData, 174
- nxtrue
  - amimodel, 49
  - Model, 142
  - ReturnData, 174
- ny
  - amidata, 17
  - amimodel, 49
  - Model, 142
  - ReturnData, 174
- nytrue
  - amimodel, 49
  - Model, 142
  - ReturnData, 174
- nz
  - amidata, 18
  - amimodel, 51
  - Model, 142
  - ReturnData, 174
- nztrue
  - amimodel, 51
  - Model, 143
  - ReturnData, 175
- o2flag
  - amimodel, 55
- o2mode
  - Model, 144
  - ReturnData, 176
- observable
  - SBMLode, 185
- observable\_name

- SBMLode, 185
- optsym, 154
  - getoptimized, 155
  - optsym, 155
- order
  - ReturnData, 171
- ordering
  - amioption, 63
  - UserData, 226
- outputcount
  - amised, 69
- p
  - TempData, 202
  - UserData, 221
- param
  - amimodel, 54
  - SBMLode, 186
- parameter
  - SBMLode, 186
- pbar
  - amioption, 61
  - UserData, 222
- plist
  - UserData, 220
- pnom
  - SBMLode, 189
- prepDataSensis
  - ForwardProblem, 89
- prepEventSensis
  - ForwardProblem, 83
- print
  - UserData, 219
- printErrMsgIdAndTxt
  - amici.cpp, 231
- printLocalVars
  - amifun, 25
- printWarnMsgIdAndTxt
  - amici.cpp, 232
- pscale
  - amioption, 65
  - ReturnData, 176
  - UserData, 221
- qpositivex
  - amioption, 60
  - UserData, 220
- rdata
  - TempData, 217
- reaction
  - SBMLode, 186
- readOptionData
  - amici\_interface\_matlab.cpp, 237
- readOptionScalar
  - amici\_interface\_matlab.cpp, 237
- recompile
  - amimodel, 55
- ReturnData, 155
  - ~ReturnData, 158
  - applyChainRuleFactorToSimulationResults, 161
  - chi2, 172
  - copyFromUserData, 161
  - dxdotdp, 166
  - dydp, 166
  - dydx, 166
  - freeFieldsOnDestruction, 173
  - initField1, 163
  - initField2, 163
  - initField3, 164
  - initField4, 165
  - initFields, 162
  - invalidate, 159
  - J, 166
  - llh, 172
  - ne, 175
  - newton\_maxsteps, 176
  - newton\_numlinsteps, 172
  - newton\_numsteps, 171
  - newton\_status, 171
  - newton\_time, 171
  - nJ, 175
  - nk, 173
  - nmaxevent, 175
  - np, 173
  - nplist, 175
  - nt, 176
  - numerrtestfails, 170
  - numerrtestfailsB, 170
  - numnonlinsolvconvfails, 170
  - numnonlinsolvconvfailsB, 171
  - numrhsevals, 170
  - numrhsevalsB, 170
  - numsteps, 169
  - numstepsB, 169
  - nx, 174
  - nxtrue, 174
  - ny, 174
  - nytrue, 174
  - nz, 174
  - nztrue, 175
  - o2mode, 176
  - order, 171
  - pscale, 176
  - ReturnData, 158
  - rz, 167
  - s2llh, 173
  - s2rz, 168
  - sensi, 176
  - sensi\_meth, 177
  - setDefaults, 159
  - setLikelihoodSensitivityFirstOrderNaN, 159
  - setLikelihoodSensitivitySecondOrderNaN, 160
  - sigmay, 169
  - sigmaz, 167
  - sllh, 172
  - srz, 168

- ssigmay, 169
- ssigmaz, 167
- status, 173
- sx, 168
- sy, 169
- sz, 167
- ts, 165
- x, 168
- xdot, 166
- xss, 172
- y, 168
- z, 167
- ReturnDataMatlab, 177
  - initField1, 179
  - initField2, 180
  - initField3, 180
  - initField4, 181
  - initFields, 179
  - mxsol, 181
  - ReturnDataMatlab, 178
- rootidx
  - TempData, 211
- rootsfound
  - TempData, 211
- rootvals
  - TempData, 211
- rowvals
  - amimodel, 52
- rowvalsB
  - amimodel, 53
- rtol
  - amioption, 60
  - UserData, 223
- runAmiciSimulation
  - amici.cpp, 230
- rz
  - ReturnData, 167
- s2llh
  - ReturnData, 173
- s2rz
  - ReturnData, 168
- SBML2AMICI.m, 227
- SBML2AMICI, 227
- SBML2AMICI
  - SBML2AMICI.m, 227
- SBMLImporter/computeBracketLevel.m, 228
- SBMLode, 182
  - bolus, 188
  - checkODE, 184
  - compartment, 186
  - condition, 187
  - constant, 186
  - flux, 187
  - funmath, 188
  - initState, 187
  - knom, 189
  - kvolume, 187
  - observable, 185
  - observable\_name, 185
  - param, 186
  - parameter, 186
  - pnom, 189
  - reaction, 186
  - SBMLode, 183
  - state, 185
  - stoichiometry, 188
  - time\_symbol, 189
  - trigger, 188
  - volume, 187
  - writeAMICI, 185
  - xdot, 188
- sdx
  - TempData, 206
- sedml
  - amised, 68
- sens\_ind
  - amioption, 60
- sensi
  - amioption, 63
  - ReturnData, 176
  - UserData, 222
- sensi\_meth
  - amioption, 63
  - ReturnData, 177
  - UserData, 224
- sensiflag
  - amifun, 34
- setDefault
  - ExpData, 76
  - ReturnData, 159
- setHflag
  - amievent, 21
- setLikelihoodSensitivityFirstOrderNaN
  - ReturnData, 159
- setLikelihoodSensitivitySecondOrderNaN
  - ReturnData, 160
- setupAMIB
  - Solver, 192
- setupAMI
  - Solver, 191
- seval
  - spline.cpp, 244
- Sigma\_Y
  - amidata, 19
- Sigma\_Z
  - amidata, 19
- sigmay
  - ExpData, 77
  - ReturnData, 169
  - TempData, 210
- sigmaz
  - ExpData, 77
  - ReturnData, 167
  - TempData, 210
- sign
  - symbolic\_functions.cpp, 251

- sinteg
  - spline.cpp, 246
- slh
  - ReturnData, 172
- Solver, 189
  - ami\_mem, 194
  - getDiagnosis, 193
  - getDiagnosisB, 193
  - setupAMIB, 192
  - setupAMI, 191
- solver
  - TempData, 217
- sparseidx
  - amimodel, 52
- sparseidxB
  - amimodel, 53
- spline
  - spline.cpp, 243
- spline.cpp
  - seval, 244
  - sinteg, 246
  - spline, 243
- splineflag
  - amimodel, 56
- src/amici.cpp, 229
- src/amici\_interface\_cpp.cpp, 232
- src/amici\_interface\_matlab.cpp, 236
- src/spline.cpp, 242
- src/symbolic\_functions.cpp, 247
- srz
  - ReturnData, 168
- ss
  - amioption, 64
- ssigmay
  - ReturnData, 169
- ssigmaz
  - ReturnData, 167
- state
  - SBMLode, 185
- status
  - ReturnData, 173
- stau
  - TempData, 215
- steadystate
  - amimodel, 48
- SteadystateProblem, 194
  - applyNewtonsMethod, 196
  - getNewtonOutput, 197
  - getNewtonSimulation, 198
  - linsolveSPBCG, 198
  - workSteadyStateProblem, 195
- stldet
  - amioption, 62
  - UserData, 225
- stoichiometry
  - SBMLode, 188
- storeJacobianAndDerivativeInReturnData
  - ForwardProblem, 81
- strsym
  - amifun, 32
- strsym\_old
  - amifun, 32
- sx
  - ReturnData, 168
  - TempData, 205
- sx0
  - amioption, 64
- sx0data
  - UserData, 225
- sy
  - ReturnData, 169
- sym
  - amifun, 32
  - amimodel, 46
- sym\_noopt
  - amifun, 32
- symbolic/am\_and.m, 259
- symbolic/am\_eq.m, 260
- symbolic/am\_ge.m, 261
- symbolic/am\_gt.m, 262
- symbolic/am\_if.m, 263
- symbolic/am\_le.m, 264
- symbolic/am\_lt.m, 265
- symbolic/am\_max.m, 266
- symbolic/am\_min.m, 266
- symbolic/am\_or.m, 267
- symbolic/am\_pieewise.m, 268
- symbolic/am\_stepfun.m, 269
- symbolic/am\_xor.m, 270
- symbolic\_functions.cpp
  - am\_DDspline, 258
  - am\_DDspline\_pos, 258
  - am\_Dspline, 256
  - am\_Dspline\_pos, 257
  - am\_max, 253
  - am\_min, 251
  - am\_spline, 255
  - am\_spline\_pos, 255
  - amiGetNaN, 249
  - amilsInf, 248
  - amilsNaN, 248
  - amilog, 249
  - Dam\_max, 254
  - Dam\_min, 252
  - dirac, 250
  - heaviside, 250
  - sign, 251
- sz
  - ReturnData, 167
- t
  - amidata, 18
  - TempData, 202
- t0
  - amimodel, 48
- TempData, 200
  - dJrzdsigma, 208

- dJrzd, [208](#)
- dJydp, [206](#)
- dJydsigma, [207](#)
- dJydx, [207](#)
- dJydy, [207](#)
- dJzdp, [207](#)
- dJzdsigma, [208](#)
- dJzdx, [208](#)
- dJzdz, [208](#)
- deltaqB, [213](#)
- deltasx, [212](#)
- deltax, [212](#)
- deltaxB, [212](#)
- dfdx, [215](#)
- discs, [213](#)
- drzdp, [209](#)
- drzdx, [209](#)
- dsigmaydp, [210](#)
- dsigmazdp, [211](#)
- dwdp, [214](#)
- dwdx, [214](#)
- dx, [204](#)
- dx\_old, [204](#)
- dxB, [205](#)
- dxdotdp, [214](#)
- dydp, [209](#)
- dydx, [210](#)
- dzdp, [209](#)
- dzdx, [209](#)
- h, [212](#)
- h\_udata, [212](#)
- irdiscs, [213](#)
- iroot, [215](#)
- J, [213](#)
- Jtmp, [206](#)
- Jy, [206](#)
- Jz, [207](#)
- llhS0, [206](#)
- M, [214](#)
- model, [217](#)
- nan\_JDiag, [216](#)
- nan\_JSparse, [216](#)
- nan\_dxdotdp, [215](#)
- nan\_J, [216](#)
- nan\_qBdot, [217](#)
- nan\_xBdot, [216](#)
- nan\_xdot, [216](#)
- np1ist, [215](#)
- nroots, [211](#)
- p, [202](#)
- rdata, [217](#)
- rootidx, [211](#)
- rootsfound, [211](#)
- rootvals, [211](#)
- sdx, [206](#)
- sigmay, [210](#)
- sigmaz, [210](#)
- solver, [217](#)
- stau, [215](#)
- sx, [205](#)
- t, [202](#)
- TempData, [202](#)
- udata, [217](#)
- w, [214](#)
- which, [213](#)
- x, [203](#)
- x\_disc, [203](#)
- x\_old, [203](#)
- xB\_old, [205](#)
- xQB\_old, [205](#)
- xQB, [205](#)
- xB, [204](#)
- xdot, [204](#)
- xdot\_disc, [203](#)
- xdot\_old, [204](#)
- xdot\_old\_disc, [203](#)
- yS0, [210](#)
- tests/cpputest/createTestingData.m, [271](#)
- tests/cpputest/wrapTestModels.m, [271](#)
- time\_symbol
  - SBMLode, [189](#)
- trigger
  - amievent, [22](#)
  - SBMLode, [188](#)
- ts
  - ReturnData, [165](#)
  - UserData, [222](#)
- tstart
  - amioption, [61](#)
  - UserData, [221](#)
- ubw
  - amimodel, [51](#)
  - Model, [144](#)
- udata
  - TempData, [217](#)
- unscaleParameters
  - UserData, [219](#)
- updateHeaviside
  - ForwardProblem, [95](#)
- updateHeavisideB
  - BackwardProblem, [73](#)
- updateModelName
  - amimodel, [39](#)
- updateRHS
  - amimodel, [38](#)
- UserData, [218](#)
  - atol, [222](#)
  - init, [219](#)
  - interpType, [224](#)
  - ism, [224](#)
  - iter, [225](#)
  - k, [221](#)
  - linsol, [224](#)
  - Imm, [225](#)
  - maxsteps, [223](#)
  - newton\_maxlinsteps, [223](#)

- newton\_maxsteps, [223](#)
- newton\_precon, [224](#)
- newton\_preeq, [223](#)
- nmaxevent, [220](#)
- nplist, [220](#)
- nt, [221](#)
- ordering, [226](#)
- p, [221](#)
- pbar, [222](#)
- plist, [220](#)
- print, [219](#)
- pscale, [221](#)
- qpositivex, [220](#)
- rtol, [223](#)
- sensi, [222](#)
- sensi\_meth, [224](#)
- stldet, [225](#)
- sx0data, [225](#)
- ts, [222](#)
- tstart, [221](#)
- unscaleParameters, [219](#)
- x0data, [225](#)
- xbar, [222](#)
- userDataFromMatlabCall
  - amici\_interface\_matlab.cpp, [238](#)
- varidx
  - amised, [69](#)
- varsym
  - amised, [69](#)
- volume
  - SBMLode, [187](#)
- w
  - TempData, [214](#)
- which
  - TempData, [213](#)
- workBackwardProblem
  - BackwardProblem, [70](#)
- workForwardProblem
  - ForwardProblem, [78](#)
- workSteadyStateProblem
  - SteadystateProblem, [195](#)
- wrap\_path
  - amimodel, [55](#)
- writeAMICI
  - SBMLode, [185](#)
- writeCcode
  - amifun, [26](#)
- writeCcode\_sensi
  - amifun, [26](#)
- writeMcode
  - amifun, [27](#)
- wtype
  - amimodel, [48](#)
- x
  - ReturnData, [168](#)
  - TempData, [203](#)
- x0
  - amioption, [64](#)
- x0data
  - UserData, [225](#)
- x\_disc
  - TempData, [203](#)
- x\_old
  - TempData, [203](#)
- xB\_old
  - TempData, [205](#)
- xQB\_old
  - TempData, [205](#)
- xQB
  - TempData, [205](#)
- xB
  - TempData, [204](#)
- xbar
  - UserData, [222](#)
- xdot
  - ReturnData, [166](#)
  - SBMLode, [188](#)
  - TempData, [204](#)
- xdot\_disc
  - TempData, [203](#)
- xdot\_old
  - TempData, [204](#)
- xdot\_old\_disc
  - TempData, [203](#)
- xss
  - ReturnData, [172](#)
- Y
  - amidata, [19](#)
- y
  - ReturnData, [168](#)
- yS0
  - TempData, [210](#)
- Z
  - amidata, [19](#)
- z
  - amievent, [22](#)
  - ReturnData, [167](#)
- z2event
  - amimodel, [56](#)
  - amioption, [65](#)
  - Model, [145](#)