AMICI

Generated by Doxygen 1.8.14

Contents

1 About AMICI **2**

2 License Conditions **2**

3 How to contribute **3**

4 Installation **3**

5 Python Interface **5**

6 MATLAB Interface **7**

7 C++ Interface **13**

8 FAQ **14**

9 Namespace Documentation **14**

9.1 amici Namespace Reference 14

9.2 amici.sbml_import Namespace Reference 66

10 Class Documentation **70**

10.1 AmiException Class Reference 70

10.2 AmiVector Class Reference 73

10.3 AmiVectorArray Class Reference 80

10.4 BackwardProblem Class Reference 87

10.5 CvodeException Class Reference 93

10.6 ExpData Class Reference 94

10.7 ForwardProblem Class Reference 100

10.8 IDAException Class Reference 110

10.9 IntegrationFailure Class Reference 111

10.10 Model Class Reference 113

10.11 Model_DAE Class Reference 214

10.12 Model_ODE Class Reference 242

10.13 NewtonFailure Class Reference 270

10.14 NewtonSolver Class Reference . . . 271
10.15 NewtonSolverDense Class Reference . . . 279
10.16 NewtonSolverIterative Class Reference . . . 281
10.17 NewtonSolverSparse Class Reference . . . 285
10.18 ReturnData Class Reference . . . 288
10.19 SBMLException Class Reference . . . 305
10.20 SbmlImporter Class Reference . . . 305
10.21 TemplateAmici Class Reference . . . 336
10.22 SetupFailure Class Reference . . . 336
10.23 Solver Class Reference . . . 337
10.24 SteadystateProblem Class Reference . . . 398
10.25 amidata Class Reference . . . 402
10.26 amievent Class Reference . . . 407
10.27 amifun Class Reference . . . 410
10.28 amimodel Class Reference . . . 417
10.29 amioption Class Reference . . . 441
10.30 amised Class Reference . . . 449
10.31 optsym Class Reference . . . 453

11 File Documentation **454**

11.1 am_and.m File Reference . . . 454
11.2 am_eq.m File Reference . . . 455
11.3 am_ge.m File Reference . . . 456
11.4 am_gt.m File Reference . . . 457
11.5 am_if.m File Reference . . . 458
11.6 am_le.m File Reference . . . 459
11.7 am_lt.m File Reference . . . 460
11.8 am_max.m File Reference . . . 461
11.9 am_min.m File Reference . . . 462
11.10 am_or.m File Reference . . . 463
11.11 am_piecewise.m File Reference . . . 464
11.12 am_stepfun.m File Reference . . . 465
11.13 am_xor.m File Reference . . . 466
11.14 amici.cpp File Reference . . . 467
11.15 amiwrap.m File Reference . . . 468
11.16 cblas.cpp File Reference . . . 469
11.17 interface_matlab.cpp File Reference . . . 470
11.18 SBML2AMICI.m File Reference . . . 472
11.19 spline.cpp File Reference . . . 473
11.20 symbolic_functions.cpp File Reference . . . 473

Index 475

1 About AMICI

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

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

Current build status

Linux/Mac: `![TravisCI][TravisBade]`

Coverage: `![Coverage Status]`

2 License Conditions

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

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

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

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

3 How to contribute

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

Making code changes

When making code changes:

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

Adding/Updating tests

To add new tests add a new corresponding python script (see, e.g., `tests/example_dirac.py`) and add it to and run `tests/generateTestConfigurationForExamples.sh` To update test results replace `tests/cpputest/expectedResults.h5` by `tests/cpputest/writeResults.h5.bak` [ONLY DO THIS AFTER TRIPLE CHECKING CORRECTNESS OF RESULTS]

4 Installation

Availability

The sources for AMICI are accessible as

- Source `tarball`
- Source `zip`
- GIT repository on `github`

Obtaining AMICI via the GIT versioning system

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

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

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

Installation

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

MATLAB

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

```
installAMICI.m
```

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

```
savepath
```

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

```
mex -setup c
```

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

Python

To use AMICI from Python, run the

```
./scripts/buildSundials.sh
./scripts/buildSuitesparse.sh
./scripts/buildAmici.sh
```

scripts to compile amici and generate the python module. After successful execution of the script, amici is available as system wide module and can be imported via

```
import amici
```

C++

To use AMICI from C++, run the

```
./scripts/buildSundials.sh
./scripts/buildSuitesparse.sh
./scripts/buildAmici.sh
```

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

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

Dependencies

The MATLAB interface requires the Mathworks Symbolic Toolbox for model generation via `amiwrap(...)`, but not for execution of precompiled models.

The Python and C++ interfaces require `cmake` and `cblas` to be installed.

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

AMICI uses the following packages from SUNDIALS:

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

IDAS

AMICI uses the following packages from SuiteSparse:

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

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

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

5 Python Interface

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

Model Definition

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

SBML input

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

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

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

```
sbml = sbmlImporter.sbml
```

Constants

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

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

Observables

assignment rules that should be considered as observables can extracted using the `amici.assignment↩Rules2observables` function

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

Standard Deviations

standard deviations can be specified as dictionaries ...

```
sigmas = {'observable_x1withsigma': 'observable_x1withsigma_sigma'}
```

Model Compilation

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

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

Model Simulation

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

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

the compiled model can now be imported as python module

```
import test as modelModule
```

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

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

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

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

for simulation we need to generate a solver instance

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

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

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

6 MATLAB Interface

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

Model Definition

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

Header

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

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

Options

Set the options by specifying the respective field of the modelstruct

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

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

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

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

States

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

```
syms state1 state2 state3
```

Create the state vector containing all states:

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

Parameters

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

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

Create the parameters vector

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

Constants

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

```
syms const1 const2
```

Create the parameters vector

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

Differential Equation

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

```
syms t
```

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

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

or

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

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

For DAEs also specify the mass matrix.

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

The specification of M may depend on parameters and constants.

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

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

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

Initial Conditions

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

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

Observables

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

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

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

Events

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

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

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

Standard Deviation

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

Standard deviaton for observable data is denoted by sigma_y

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

Standard deviaton for event data is denoted by sigma_t

```
model.syms.sigma_t(1) = param6;
```

Both `sigma_y` and `sigma_t` can either be a scalar or of the same dimension as the observables / events function. They can depend on time and parameters but must not depend on the states or observables. The values provided in `sigma_y` and `sigma_t` will only be used if the value in `D.Sigma_Y` or `D.Sigma_T` in the user-provided data struct is `NaN`. See simulation for details.

Objective Function

By default, AMICI assume a normal noise model and use the corresponding negative log-likelihood

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

as objective function.

SBML

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

Model Compilation

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

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

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

Model Simulation

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

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

Integration

Define a time vector:

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

Generate a parameter vector:

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

Generate a constants vector:

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

Integrate:

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

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

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

Forward Sensitivities

Set the sensitivity computation to forward sensitivities and Integrate:

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

The integration status will be indicated by the `sol.status` flag. Negative values 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 event outputs will then be available as `sol.z`, with the derivative with respect to the parameters in `sol.sz`. If no event occured there will be an event at the end of the considered interval with the final value of the root function stored in `sol.rz`, with the derivative with respect to the parameters in `sol.srz`.

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.

Adjoint Sensitivities

Set the sensitivity computation to adjoint sensitivities:

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

Define Experimental Data:

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

The `NaN` values in `Sigma_Y` and `Sigma_T` will be replaced by the specification in `model.syms.sigma_y` and `model.syms.sigma_t`. Data points with `NaN` value will be completely ignored.

Integrate:

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

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

Steady State Sensitivities

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

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

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

```
t = Inf;
```

Set the sensitivity computation to steady state sensitivities:

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

Integrate:

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

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

7 C++ Interface

The Python Interface and MATLAB Interface can translate the model definition into C++ code, which is then compiled into a .mex file or a python module. Advanced users can also use this code within stand-alone C/C++ application for use in other environments (e.g. on high performance computing systems). This section will give a short overview over the generated files and provide a brief introduction of how this code can be included in other applications.

Generated model files

`amiwrap.m` and `amici.SbmlImporter.sbml2amici` write the model source files to ${AMICI_ROOT_DIR}/models/${MODEL_NAME} by default. The content of a model source directory might look something like this (given `MODEL_NAME=model_steadystate`):

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

Running a simulation

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

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

Compiling and linking

The complete AMICI API is available through `amici.h`; this is the only header file that needs to be included. `hdf5.h` provides some functions for reading and writing `HDF5` files).

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

Along with `main.cpp`, a `CMake` file (`CMakeLists.txt`) will be generated automatically. The CMake file shows the abovementioned library dependencies. These files provide a scaffold for a standalone simulation program. The required numerical libraries are shipped with AMICI. To compile them, run `${AMICI_ROOT_DIR}/scripts/run-tests.sh` once. HDF5 libraries and header files need to be installed separately. More information on how to run the compiled program is provided in `main.cpp`.

8 FAQ

Q: My model fails to build.

A: Remove the corresponding model directory located in AMICI/models/∗yourmodelname∗ and compile again.

Q: It still does not compile.

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

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

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

Q: The simulation/sensitivities I get are incorrect.

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

9 Namespace Documentation

9.1 amici Namespace Reference

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

Namespaces

- sbml_import

  The python sbml import module for python.

Classes

- class AmiException

  amici exception handler class
- class AmiVector
- class AmiVectorArray
- class BackwardProblem

  class to solve backwards problems.
- class CvodeException

  cvode exception handler class
- class ExpData

  ExpData carries all information about experimental or condition-specific data.
- class ForwardProblem

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

  ida exception handler class
- class IntegrationFailure

  integration failure exception this exception should be thrown when an integration failure occured for this exception we can assume that we can recover from the exception and return a solution struct to the user
- class Model

The Model class represents an AMICI ODE model. The model can compute various model related quantities based on symbolically generated code.

- class Model_DAE

  The Model class represents an AMICI DAE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchroneously.

- class Model_ODE

  The Model class represents an AMICI ODE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchroneously.

- class NewtonFailure

  newton failure exception this exception should be thrown when the steady state computation failed to converge for this exception we can assume that we can recover from the exception and return a solution struct to the user

- class NewtonSolver

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

- class NewtonSolverDense

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

- class NewtonSolverIterative

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

- class NewtonSolverSparse

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

- class ReturnData

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

- class SetupFailure

  setup failure exception this exception should be thrown when the solver setup failed for this exception we can assume that we cannot recover from the exception and an error will be thrown

- class Solver
- class SteadystateProblem

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

Typedefs

- typedef double realtype
- typedef void(∗ msgIdAndTxtFp) (const char ∗identifier, const char ∗format,...)

  msgIdAndTxtFp

Enumerations

- enum AMICI_BLAS_LAYOUT { **AMICI_BLAS_RowMajor** = 101, **AMICI_BLAS_ColMajor** = 102 }
- enum AMICI_BLAS_TRANSPOSE { **AMICI_BLAS_NoTrans** = 111, **AMICI_BLAS_Trans** = 112, **AMICI_BLAS_ConjTrans** = 113 }
- enum AMICI_parameter_scaling { **AMICI_SCALING_NONE**, **AMICI_SCALING_LN**, **AMICI_SCALING_LOG10** }
- enum AMICI_o2mode { **AMICI_O2MODE_NONE**, **AMICI_O2MODE_FULL**, **AMICI_O2MODE_DIR** }
- enum AMICI_sensi_order { **AMICI_SENSI_ORDER_NONE**, **AMICI_SENSI_ORDER_FIRST**, **AMICI_SENSI_ORDER_SECOND** }
- enum AMICI_sensi_meth { **AMICI_SENSI_NONE**, **AMICI_SENSI_FSA**, **AMICI_SENSI_ASA**, **AMICI_SENSI_SS** }
- enum LinearSolver {
  AMICI_DENSE = 1, **AMICI_BAND** = 2, **AMICI_LAPACKDENSE** = 3, **AMICI_LAPACKBAND** = 4,
  AMICI_DIAG = 5, **AMICI_SPGMR** = 6, **AMICI_SPBCG** = 7, **AMICI_SPTFQMR** = 8,
  AMICI_KLU = 9 }
- enum InternalSensitivityMethod { **SIMULTANEOUS** = 1, **STAGGERED** = 2, **STAGGERED1** = 3 }

- enum InterpolationType { **HERMITE** = 1, **POLYNOMIAL** = 2 }
- enum LinearMultistepMethod { **ADAMS** = 1, **BDF** = 2 }
- enum NonlinearSolverIteration { **FUNCTIONAL** = 1, **NEWTON** = 2 }
- enum StateOrdering { **AMD**, **COLAMD**, **natural** }
- enum mexRhsArguments {
  RHS_TIMEPOINTS, **RHS_PARAMETERS**, **RHS_CONSTANTS**, **RHS_OPTIONS**, **RHS_PLIST**, **RHS_XSCALE_UNUSED**, **RHS_INITIALIZATION**, **RHS_DATA**, **RHS_NUMARGS_REQUIRED** = RHS_DATA, **RHS_NUMARGS** }

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

Functions

- void printErrMsgIdAndTxt (const char ∗identifier, const char ∗format,...)
- void printWarnMsgIdAndTxt (const char ∗identifier, const char ∗format,...)
- std::unique_ptr< ReturnData > runAmiciSimulation (Solver &solver, const ExpData ∗edata, Model &model)
- void amici_dgemv (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)
- 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 setModelData (const mxArray ∗prhs[ ], int nrhs, Model &model)

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

  setSolverOptions solver options from the matlab call to a solver object
- ReturnDataMatlab ∗ setupReturnData (mxArray ∗plhs[ ], int nlhs)
- ExpData ∗ expDataFromMatlabCall (const mxArray ∗prhs[ ], const Model &model)
- int checkFinite (const int N, const realtype ∗array, const char ∗fun)
- bool operator== (const Model &a, const Model &b)
- mxArray ∗ getReturnDataMatlabFromAmiciCall (ReturnData const ∗rdata)
- mxArray ∗ initMatlabReturnFields (ReturnData const ∗rdata)
- mxArray ∗ initMatlabDiagnosisFields (ReturnData const ∗rdata)
- template<typename T >
  void writeMatlabField0 (mxArray ∗matlabStruct, const char ∗fieldName, T fieldData)
- template<typename T >
  void writeMatlabField1 (mxArray ∗matlabStruct, const char ∗fieldName, std::vector< T > fieldData, const int dim0)
- template<typename T >
  void writeMatlabField2 (mxArray ∗matlabStruct, const char ∗fieldName, std::vector< T > fieldData, int dim0, int dim1, std::vector< int > perm)
- template<typename T >
  void writeMatlabField3 (mxArray ∗matlabStruct, const char ∗fieldName, std::vector< T > fieldData, int dim0, int dim1, int dim2, std::vector< int > perm)
- template<typename T >
  void writeMatlabField4 (mxArray ∗matlabStruct, const char ∗fieldName, std::vector< T > fieldData, int dim0, int dim1, int dim2, int dim3, std::vector< int > perm)
- double ∗ initAndAttachArray (mxArray ∗matlabStruct, const char ∗fieldName, std::vector< mwSize > dim)
- void checkFieldNames (const char ∗∗fieldNames, const int fieldCount)
- template<typename T >
  std::vector< T > reorder (const std::vector< T > input, const std::vector< int > order)
- template<typename T >
  char ∗ serializeToChar (T const &data, int ∗size)
- template<typename T >
  T deserializeFromChar (const char ∗buffer, int size)

- template<typename T >
  std::string serializeToString (T const &data)
- template<typename T >
  std::vector< char > serializeToStdVec (T const &data)
- template<typename T >
  T deserializeFromString (std::string const &serialized)
- bool operator== (const Solver &a, const Solver &b)
- int spline (int n, int end1, int end2, double slope1, double slope2, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])
- double seval (int n, double u, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])

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

  Convenience wrapper around amici.runAmiciSimulation (generated by swig)
- def rdataToNumPyArrays (rdata)

  Convenience wrapper ReturnData class (generated by swig)
- def getFieldAsNumPyArray (rdata, field)

  Convert ReturnData field to numpy array with dimensions according to model dimensions in rdata.
- int dbl2int (const double x)
- char amici_blasCBlasTransToBlasTrans (AMICI_BLAS_TRANSPOSE trans)

Variables

- msgIdAndTxtFp errMsgIdAndTxt = &printErrMsgIdAndTxt
- msgIdAndTxtFp warnMsgIdAndTxt = &printWarnMsgIdAndTxt
- constexpr double pi = M_PI
- def dirname = os.path.dirname(__file__)

  absolute path to parent directory of this file
- amici_path

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

  absolute path of the amici swig directory
- amiciSrcPath = os.path.join(amici_path, 'src')

  absolute path of the amici source directory
- def amiciModulePath = os.path.dirname(__file__)

  absolute root path of the amici module

9.1.1 Detailed Description

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

Getting started:

```
 creating a extension module for an SBML model:
import amici
amiSbml = amici.SbmlImporter('mymodel.sbml')
amiSbml.sbml2amici('modelName', 'outputDirectory')

 using the created module (set python path)
import modelName
help(modelName)
```

9.1.2 Typedef Documentation

9.1.2.1 realtype

```
typedef double realtype
```

defines variable type for simulation variables (determines numerical accuracy)

Definition at line 50 of file defines.h.

9.1.2.2 msgIdAndTxtFp

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

Parameters

identifier	string with error message identifier
format	string with error message printf-style format
...	arguments to be formatted

Definition at line 145 of file defines.h.

9.1.3 Enumeration Type Documentation

9.1.3.1 AMICI_BLAS_LAYOUT

```
enum AMICI_BLAS_LAYOUT
```

BLAS Matrix Layout, affects dgemm and gemv calls

Definition at line 53 of file defines.h.

9.1.3.2 AMICI_BLAS_TRANSPOSE

`enum AMICI_BLAS_TRANSPOSE`

BLAS Matrix Transposion, affects dgemm and gemv calls

Definition at line 59 of file defines.h.

9.1.3.3 AMICI_parameter_scaling

`enum AMICI_parameter_scaling`

modes for parameter transformations

Definition at line 66 of file defines.h.

9.1.3.4 AMICI_o2mode

`enum AMICI_o2mode`

modes for second order sensitivity analysis

Definition at line 73 of file defines.h.

9.1.3.5 AMICI_sensi_order

`enum AMICI_sensi_order`

orders of sensitivity analysis

Definition at line 80 of file defines.h.

9.1.3.6 AMICI_sensi_meth

`enum AMICI_sensi_meth`

methods for sensitivity computation

Definition at line 87 of file defines.h.

9.1.3.7 LinearSolver

```
enum LinearSolver
```

linear solvers for CVODES/IDAS

Definition at line 95 of file defines.h.

9.1.3.8 InternalSensitivityMethod

```
enum InternalSensitivityMethod
```

CVODES/IDAS forward sensitivity computation method

Definition at line 108 of file defines.h.

9.1.3.9 InterpolationType

```
enum InterpolationType
```

CVODES/IDAS state interpolation for adjoint sensitivity analysis

Definition at line 115 of file defines.h.

9.1.3.10 LinearMultistepMethod

```
enum LinearMultistepMethod
```

CVODES/IDAS linear multistep method

Definition at line 121 of file defines.h.

9.1.3.11 NonlinearSolverIteration

```
enum NonlinearSolverIteration
```

CVODES/IDAS Nonlinear Iteration method

Definition at line 127 of file defines.h.

9.1.3.12 StateOrdering

```
enum StateOrdering
```

KLU state reordering

Definition at line 133 of file defines.h.

9.1.4 Function Documentation

9.1.4.1 printErrMsgIdAndTxt()

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

printErrMsgIdAndTxt prints a specified error message associated to the specified identifier

Parameters

in	*identifier*	error identifier **Type**: char
in	*format*	string with error message printf-style format
	...	arguments to be formatted

Returns

void

Definition at line 105 of file amici.cpp.

9.1.4.2 printWarnMsgIdAndTxt()

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

printErrMsgIdAndTxt prints a specified warning message associated to the specified identifier

Parameters

in	*identifier*	warning identifier **Type**: char
in	*format*	string with error message printf-style format
	...	arguments to be formatted

Returns

void

Definition at line 126 of file amici.cpp.

9.1.4.3 runAmiciSimulation() [1/2]

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

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

Parameters

solver	Solver instance
edata	pointer to experimental data object
model	Model specification object

Returns

rdata pointer to return data object

Definition at line 59 of file amici.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.4 amici_dgemv()

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

```
double * Y,
const int incY )
```

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

Parameters

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

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

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

Returns

void

Definition at line 73 of file cblas.cpp.

9.1.4.5 amici_dgemm()

```
void amici_dgemm (
            AMICI_BLAS_LAYOUT layout,
            AMICI_BLAS_TRANSPOSE TransA,
            AMICI_BLAS_TRANSPOSE TransB,
            const int M,
            const int N,
            const int K,
            const double alpha,
```

```
const double * A,
const int lda,
const double * B,
const int ldb,
const double beta,
double * C,
const int ldc )
```

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

Parameters

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

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

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

Returns

void

Definition at line 44 of file cblas.cpp.

Here is the caller graph for this function:

9.1.4.6 setModelData()

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

Parameters

in	*prhs*	pointer to the array of input arguments **Type**: mxArray
in	*nrhs*	number of elements in prhs
in,out	*model*	model to update

Definition at line 368 of file interface_matlab.cpp.

Here is the call graph for this function:

9.1.4.7 setSolverOptions()

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

Parameters

in	*prhs*	pointer to the array of input arguments **Type**: mxArray
in	*nrhs*	number of elements in prhs
in,out	*solver*	solver to update

Definition at line 291 of file interface_matlab.cpp.

Here is the call graph for this function:

9.1.4.8 setupReturnData()

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

setupReturnData initialises the return data struct

Parameters

in	*plhs*	user input **Type**: mxArray
in	*nlhs*	number of elements in plhs **Type**: mxArray

Returns

rdata: return data struct
Type: ∗ReturnData

9.1.4.9 expDataFromMatlabCall()

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

expDataFromMatlabCall initialises the experimental data struct

Parameters

in	*prhs*	user input **Type**: ∗mxArray

Returns

edata: experimental data struct
Type: ∗ExpData

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

Parameters

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

Returns

edata pointer to experimental data object

Definition at line 160 of file interface_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.10 checkFinite()

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

Checks the values in an array for NaNs and Infs

Parameters

N	number of elements in array
array	array
fun	name of calling function

Returns

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

Definition at line 17 of file misc.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.11 operator==() [1/2]

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

Parameters

a	
b	

Returns

Definition at line 923 of file model.cpp.

9.1.4.12 getReturnDataMatlabFromAmiciCall()

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

generates matlab mxArray from a ReturnData object

Parameters

rdata	ReturnDataObject

Returns

rdatamatlab ReturnDataObject stored as matlab compatible data

generates matlab mxArray from a ReturnData object

Parameters

rdata	ReturnDataObject

Returns

rdatamatlab ReturnDataObject stored as matlab compatible data

Definition at line 7 of file returndata_matlab.cpp.

Here is the call graph for this function:

9.1.4.13 initMatlabReturnFields()

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

allocates and initialises solution mxArray with the corresponding fields

Parameters

rdata	ReturnDataObject

Returns

Solution mxArray

allocates and initialises solution mxArray with the corresponding fields

Parameters

rdata	ReturnDataObject

Returns

Solution mxArray

Definition at line 18 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.14 initMatlabDiagnosisFields()

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

allocates and initialises diagnosis mxArray with the corresponding fields

Parameters

rdata	ReturnDataObject

Returns

Diagnosis mxArray

allocates and initialises diagnosis mxArray with the corresponding fields

Parameters

rdata	ReturnDataObject

Returns

Diagnosis mxArray

Definition at line 115 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.15 writeMatlabField0()

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

initialise vector and attach to the field

Parameters

matlabStruct	pointer of the field to which the vector will be attached
fieldName	Name of the field to which the vector will be attached
fieldData	Data wich will be stored in the field

initialise vector and attach to the field

Parameters

matlabStruct	pointer of the field to which the vector will be attached
fieldName	Name of the field to which the vector will be attached
fieldData	Data wich will be stored in the field

Definition at line 175 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.16 writeMatlabField1()

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

initialise vector and attach to the field

Parameters

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

initialise vector and attach to the field

Parameters

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

Definition at line 193 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.17 writeMatlabField2()

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

initialise matrix, attach to the field and write data

Parameters

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

initialise matrix, attach to the field and write data

Parameters

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

Definition at line 215 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.18 writeMatlabField3()

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

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

Parameters

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

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

Parameters

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

Definition at line 248 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.19 writeMatlabField4()

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

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

Parameters

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

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

Parameters

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

Definition at line 284 of file returndata_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.20 initAndAttachArray()

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

initialises the field fieldName in matlabStruct with dimension dim

Parameters

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

Returns

Pointer to field data

initialises the field fieldName in matlabStruct with dimension dim

Parameters

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

Returns

Pointer to field data

Definition at line 322 of file returndata_matlab.cpp.

Here is the caller graph for this function:

9.1.4.21 checkFieldNames()

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

checks whether fieldNames was properly allocated

Parameters

fieldNames	array of field names
fieldCount	expected number of fields in fieldNames

checks whether fieldNames was properly allocated

Parameters

fieldNames	array of field names
fieldCount	expected number of fields in fieldNames

Definition at line 343 of file returndata_matlab.cpp.

Here is the caller graph for this function:

9.1.4.22 reorder()

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

template function that reorders elements in a std::vector

Parameters

input	unordered vector
order	dimension permutation

Returns

Reordered vector

template function that reorders elements in a std::vector

Parameters

input	unordered vector
order	dimension permutation

Returns

Reordered vector

Definition at line 356 of file returndata_matlab.cpp.

Here is the caller graph for this function:

9.1.4.23 serializeToChar()

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

Serialize object to char array

Parameters

data	input object
size	maximum char length

Returns

The object serialized as char

Definition at line 161 of file serialization.h.

9.1.4.24 deserializeFromChar()

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

Deserialize object that has been serialized using serializeToChar

Parameters

buffer	serialized object
size	length of buffer

Returns

The deserialized object

Definition at line 190 of file serialization.h.

9.1.4.25 serializeToString()

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

Serialize object to string

Parameters

data	input object

Returns

The object serialized as string

Definition at line 211 of file serialization.h.

9.1.4.26 serializeToStdVec()

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

Serialize object to std::vector<char>

Parameters

data	input object

Returns

The object serialized as std::vector<char>

Definition at line 232 of file serialization.h.

9.1.4.27 deserializeFromString()

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

Deserialize object that has been serialized using serializeToString

Parameters

serialized	serialized object

Returns

The deserialized object

Definition at line 254 of file serialization.h.

9.1.4.28 operator==() [2/2]

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

Parameters

a	
b	

Returns

Definition at line 424 of file solver.cpp.

9.1.4.29 spline() [1/2]

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

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

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

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

Parameters

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

Return values

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

Notes

- The accompanying function seval() may be used to evaluate the spline while deriv will provide the first derivative.

- Using p to denote differentiation y[i] = S(X[i]) b[i] = Sp(X[i]) c[i] = Spp(X[i])/2 d[i] = Sppp(X[i])/6 (Derivative from the right)
- Since the zero elements of the arrays ARE NOW used here, all arrays to be passed from the main program should be dimensioned at least [n]. These routines will use elements [0 .. n-1].
- Adapted from the text Forsythe, G.E., Malcolm, M.A. and Moler, C.B. (1977) "Computer Methods for Mathematical Computations" Prentice Hall
- Note that although there are only n-1 polynomial segments, n elements are requird in b, c, d. The elements b[n-1], c[n-1] and d[n-1] are set to continue the last segment past x[n-1].

Definition at line 16 of file spline.cpp.

Here is the caller graph for this function:

9.1.4.30 seval()

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

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

Parameters

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

Returns

the value of the spline function at u

Notes

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

Definition at line 195 of file spline.cpp.

Here is the caller graph for this function:

9.1.4.31 sinteg()

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

Integrate the cubic spline function

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

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

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

Parameters

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

Returns

the value of the spline function at u

Notes

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

Definition at line 256 of file spline.cpp.

9.1.4.32 log()

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

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

Parameters

x	argument

Returns

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

Definition at line 62 of file symbolic_functions.cpp.

Here is the caller graph for this function:

9.1.4.33 dirac()

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

c implementation of matlab function dirac

Parameters

x	argument

Returns

if(x==0) then INF else 0

Definition at line 77 of file symbolic_functions.cpp.

9.1.4.34 heaviside()

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

c implementation of matlab function heaviside

Parameters

x	argument

Returns

if(x>0) then 1 else 0

Definition at line 92 of file symbolic_functions.cpp.

9.1.4.35 min()

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

c implementation of matlab function min

Parameters

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

Returns

if(a < b) then a else b

Definition at line 149 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.36 Dmin()

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

parameter derivative of c implementation of matlab function max

Parameters

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

Returns

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

Definition at line 191 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.37 max()

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

c implementation of matlab function max

Parameters

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

Returns

if(a > b) then a else b

Definition at line 128 of file symbolic_functions.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.38 Dmax()

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

parameter derivative of c implementation of matlab function max

Parameters

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

Returns

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

Definition at line 164 of file symbolic_functions.cpp.

Here is the caller graph for this function:

9.1.4.39 isNaN()

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

c++ interface to the isNaN function

Parameters

what	argument

Returns

isnan(what)

Definition at line 35 of file symbolic_functions.cpp.

Here is the caller graph for this function:

9.1.4.40 isInf()

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

c++ interface to the isinf function

Parameters

what	argument

Returns

isnan(what)

Definition at line 44 of file symbolic_functions.cpp.

Here is the caller graph for this function:

9.1.4.41 getNaN()

```
double getNaN ( )
```

function returning nan

Returns

NaN

Definition at line 52 of file symbolic_functions.cpp.

Here is the caller graph for this function:

9.1.4.42 sign()

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

c implementation of matlab function sign

Parameters

x	argument

Returns

0

Definition at line 107 of file symbolic_functions.cpp.

9.1.4.43 spline() [2/2]

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

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

Parameters

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

Returns

spline(t)

Definition at line 209 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.44 spline_pos()

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

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

Parameters

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

Returns

spline(t)

Definition at line 263 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.45 Dspline()

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

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

Parameters

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

Returns

dsplinedp(t)

Definition at line 319 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.46 Dspline_pos()

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

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

Parameters

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

Returns

dsplinedp(t)

Definition at line 379 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.47 DDspline()

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

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

Parameters

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

Returns

ddspline(t)

Definition at line 449 of file symbolic_functions.cpp.

9.1.4.48 DDspline_pos()

```
double DDspline_pos (
            int id1,
```

```
            int id2,
            double t,
            int num,
             ...  )
```

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

Parameters

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

Returns

ddspline(t)

Definition at line 469 of file symbolic_functions.cpp.

Here is the call graph for this function:

9.1.4.49 runAmiciSimulation() [2/2]

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

Parameters

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

Returns

ReturnData object with simulation results

Definition at line 69 of file __init__.py.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.50 rdataToNumPyArrays()

```
def amici.rdataToNumPyArrays (
              rdata )
```

Parameters

rdata	ReturnData instance with simulation results

Returns

ReturnData object with numpy array fields

Definition at line 87 of file __init__.py.

Here is the call graph for this function:

Here is the caller graph for this function:

9.1.4.51 getFieldAsNumPyArray()

```
def amici.getFieldAsNumPyArray (
              rdata,
              field )
```

Parameters

rdata	ReturnData instance with simulation results
field	Name of field

Returns

Field Data as numpy array with dimensions according to model dimensions in rdata

Definition at line 114 of file __init__.py.

Here is the caller graph for this function:

9.1.4.52 dbl2int()

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

conversion from double to int with checking for loss of data

Parameters

x	input

Returns

int_x casted value

Definition at line 285 of file interface_matlab.cpp.

Here is the caller graph for this function:

9.1.4.53 amici_blasCBlasTransToBlasTrans()

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

amici_blasCBlasTransToBlasTrans translates AMICI_BLAS_TRANSPOSE values to CBlas readable strings

Parameters

trans	flag indicating transposition and complex conjugation

Returns

cblastrans CBlas readable CHAR indicating transposition and complex conjugation

Definition at line 52 of file interface_matlab.cpp.

9.1.5 Variable Documentation

9.1.5.1 errMsgIdAndTxt

```
msgIdAndTxtFp errMsgIdAndTxt = &printErrMsgIdAndTxt
```

errMsgIdAndTxt is a function pointer for printErrMsgIdAndTxt

Definition at line 45 of file amici.cpp.

9.1.5.2 warnMsgIdAndTxt

```
msgIdAndTxtFp warnMsgIdAndTxt = &printWarnMsgIdAndTxt
```

warnMsgIdAndTxt is a function pointer for printWarnMsgIdAndTxt

Definition at line 22 of file model.h.

9.1.5.3 pi

```
constexpr double pi = M_PI
```

pi definition from MATH_DEFINES

Definition at line 10 of file defines.h.

9.1.5.4 amici_path

```
def amici_path
```

Initial value:

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

Definition at line 44 of file __init__.py.

9.2 amici.sbml_import Namespace Reference

The python sbml import module for python.

Classes

- class SBMLException
- class SbmlImporter

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

  Template format used in AMICI (see string.template for more details).

Functions

- def applyTemplate (sourceFile, targetFile, templateData)

  Load source file, apply template substitution as provided in templateData and save as targetFile.
- def getSymbols (prefix, length)

  Get symbolic matrix with symbols prefix0..prefix(length-1).
- def getSymbolicDiagonal (matrix)

  Get symbolic matrix with diagonal of matrix `matrix`.
- def getRuleVars (rules)

  Extract free symbols in SBML rule formulas.
- def assignmentRules2observables (sbml, filter=lambda *_:True)

  Turn assignment rules into observables.

9.2.1 Detailed Description

!/usr/bin/env python3

9.2.2 Function Documentation

9.2.2.1 applyTemplate()

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

Parameters

sourceFile	relative or absolute path to template file
targetFile	relative or absolute path to output file
templateData	dictionary with template keywords to substitute (key is template veriable without TemplateAmici.delimiter)

Returns

Definition at line 1286 of file sbml_import.py.

Here is the caller graph for this function:

9.2.2.2 getSymbols()

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

Parameters

prefix	variable name
length	number of symbolic variables + 1

Returns

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

Definition at line 1305 of file sbml_import.py.

Here is the caller graph for this function:

9.2.2.3 getSymbolicDiagonal()

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

Parameters

matrix	Matrix from which to return the diagonal

Returns

A Symbolic matrix with the diagonal of `matrix`.

Exceptions

Exception	The provided matrix was not square

Definition at line 1320 of file sbml_import.py.

Here is the caller graph for this function:

9.2.2.4 getRuleVars()

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

Argumentss

rules list of rules

Returns

Vector of free symbolic variables in the formulas all provided rules

Definition at line 1339 of file sbml_import.py.

Here is the caller graph for this function:

9.2.2.5 assignmentRules2observables()

```
def amici.sbml_import.assignmentRules2observables (
              sbml,
              filter = lambda *_: True )
```

Parameters

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

Returns

A dictionary(observableName:formulaString)

Definition at line 1370 of file sbml_import.py.

10 Class Documentation

10.1 AmiException Class Reference

amici exception handler class

`#include <exception.h>`

Inheritance diagram for AmiException:

Public Member Functions

- AmiException (char const ∗fmt,...)
- AmiException (const AmiException &old)
- const char ∗ what () const throw ()
- const char ∗ getBacktrace () const
- void storeBacktrace (const int nMaxFrames)

10.1.1 Detailed Description

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

Definition at line 29 of file exception.h.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 AmiException() [1/2]

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

constructor with printf style interface

Parameters

in	*fmt*	error message with printf format
in	...	printf formating variables

Definition at line 31 of file exception.h.

Here is the call graph for this function:

10.1.2.2 AmiException() [2/2]

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

copy constructor

Parameters

old	object to copy from

Definition at line 43 of file exception.h.

10.1.3 Member Function Documentation

10.1.3.1 what()

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

override of default error message function

Returns

msg error message

Definition at line 54 of file exception.h.

Here is the caller graph for this function:

10.1.3.2 getBacktrace()

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

returns the stored backtrace

Returns

trace backtrace

Definition at line 61 of file exception.h.

Here is the caller graph for this function:

10.1.3.3 storeBacktrace()

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

stores the current backtrace

Parameters

nMaxFrames	number of frams to go back in stacktrace

Definition at line 68 of file exception.h.

Here is the caller graph for this function:

10.2 AmiVector Class Reference

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

Public Member Functions

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

10.2.1 Detailed Description

AmiVector class. provides a generic interface to the NVector_Serial struct

Definition at line 15 of file vector.h.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 AmiVector() [1/3]

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

default constructor creates an std::vector<realype> and attaches the data pointer to a newly created N_Vector↩ _Serial using N_VMake_Serial ensures that the N_Vector module does not try to deallocate the data vector when calling N_VDestroy_Serial

Parameters

length	number of elements in vector

Returns

new AmiVector instance

Definition at line 26 of file vector.h.

10.2.2.2 AmiVector() [2/3]

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

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

Parameters

rvec	vector from which the data will be copied

Returns

new AmiVector instance

Definition at line 35 of file vector.h.

10.2.2.3 AmiVector() [3/3]

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

copy constructor

Parameters

vold	vector from which the data will be copied

Definition at line 43 of file vector.h.

10.2.2.4 ~AmiVector()

```
~AmiVector ( )
```

default destructor creates an std::vector<realype> and attaches the data pointer to a newly created N_Vector_↩ Serial

Definition at line 136 of file vector.h.

10.2.3 Member Function Documentation

10.2.3.1 operator=()

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

copy-move assignment operator

Parameters

other	right hand side

Returns

left hand side

Definition at line 52 of file vector.h.

10.2.3.2 data() [1/2]

```
realtype* data ( )
```

data accessor

Returns

pointer to data array

Definition at line 63 of file vector.h.

Here is the caller graph for this function:

10.2.3.3 data() [2/2]

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

const data accessor

Returns

const pointer to data array

Definition at line 70 of file vector.h.

10.2.3.4 getNVector()

```
N_Vector getNVector ( ) const
```

N_Vector accessor

Returns

N_Vector

Definition at line 77 of file vector.h.

Here is the caller graph for this function:

10.2.3.5 getVector()

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

Vector accessor

Returns

Vector

Definition at line 84 of file vector.h.

10.2.3.6 getLength()

```
int getLength ( ) const
```

returns the length of the vector

Returns

length

Definition at line 91 of file vector.h.

10.2.3.7 reset()

```
void reset ( )
```

resets the Vector by filling with zero values

Definition at line 97 of file vector.h.

Here is the caller graph for this function:

10.2.3.8 minus()

```
void minus ( )
```

changes the sign of data elements

Definition at line 103 of file vector.h.

Here is the caller graph for this function:

10.2.3.9 set()

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

sets all data elements to a specific value

Parameters

val	value for data elements

Definition at line 112 of file vector.h.

Here is the caller graph for this function:

10.2.3.10 operator[]()

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

accessor to data elements of the vector

Parameters

pos	index of element

Returns

element

Definition at line 120 of file vector.h.

10.2.3.11 at()

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

accessor to data elements of the vector

Parameters

pos	index of element

Returns

element

Definition at line 128 of file vector.h.

Here is the caller graph for this function:

10.3 AmiVectorArray Class Reference

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

Public Member Functions

- AmiVectorArray (const long int length_inner, const long int length_outer)
- AmiVectorArray (const AmiVectorArray &vaold)
- realtype ∗ data (int pos)

- const realtype ∗ data (int pos) const
- realtype & at (int ipos, int jpos)
- N_Vector ∗ getNVectorArray ()
- N_Vector getNVector (int pos)
- AmiVector & operator[ ] (int pos)
- const AmiVector & operator[ ] (int pos) const
- int getLength () const
- void reset ()
- ∼AmiVectorArray ()

10.3.1 Detailed Description

AmiVectorArray class. provides a generic interface to arrays of NVector_Serial structs

Definition at line 152 of file vector.h.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 AmiVectorArray() [1/2]

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

default constructor creates an std::vector<realype> and attaches the data pointer to a newly created N_Vector↩ Array using CloneVectorArrayEmpty ensures that the N_Vector module does not try to deallocate the data vector when calling N_VDestroyVectorArray_Serial

Parameters

length_inner	length of vectors
length_outer	number of vectors

Returns

New AmiVectorArray instance

Definition at line 164 of file vector.h.

10.3.2.2 AmiVectorArray() [2/2]

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

copy constructor

Parameters

vaold	object to copy from

Returns

new AmiVectorArray instance

Definition at line 177 of file vector.h.

Here is the call graph for this function:

10.3.2.3 ~AmiVectorArray()

```
~AmiVectorArray ( )
```

default destructor

Definition at line 257 of file vector.h.

10.3.3 Member Function Documentation

10.3.3.1 data() [1/2]

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

accessor to data of AmiVector elements

Parameters

pos	index of AmiVector

Returns

pointer to data array

Definition at line 188 of file vector.h.

Here is the caller graph for this function:

10.3.3.2 data() [2/2]

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

const accessor to data of AmiVector elements

Parameters

pos	index of AmiVector

Returns

const pointer to data array

Definition at line 196 of file vector.h.

10.3.3.3 at()

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

accessor to elements of AmiVector elements

Parameters

ipos	inner index in AmiVector
jpos	outer index in AmiVectorArray

Returns

element

Definition at line 205 of file vector.h.

Here is the caller graph for this function:

10.3.3.4 getNVectorArray()

```
N_Vector* getNVectorArray ( )
```

accessor to NVectorArray

Returns

N_VectorArray

Definition at line 212 of file vector.h.

10.3.3.5 getNVector()

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

accessor to NVector element

Parameters

pos	index of corresponding AmiVector

Returns

N_Vector

Definition at line 220 of file vector.h.

10.3.3.6 operator[]() [1/2]

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

accessor to AmiVector elements

Parameters

pos	index of AmiVector

Returns

AmiVector

Definition at line 228 of file vector.h.

10.3.3.7 operator[]() [2/2]

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

const accessor to AmiVector elements

Parameters

pos	index of AmiVector

Returns

const AmiVector

Definition at line 236 of file vector.h.

10.3.3.8 getLength()

```
int getLength ( ) const
```

length of AmiVectorArray

Returns

length

Definition at line 243 of file vector.h.

Here is the caller graph for this function:

10.3.3.9 reset()

```
void reset ( )
```

resets every AmiVector in AmiVectorArray

Definition at line 249 of file vector.h.

Here is the caller graph for this function:

10.4 BackwardProblem Class Reference

class to solve backwards problems.

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

Public Member Functions

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

10.4.1 Detailed Description

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

Definition at line 23 of file backwardproblem.h.

10.4.2 Constructor & Destructor Documentation

10.4.2.1 BackwardProblem()

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

default constructor

Parameters

fwd	pointer to corresponding forward problem

Returns

new BackwardProblem instance

Definition at line 18 of file backwardproblem.cpp.

Here is the call graph for this function:

10.4.3 Member Function Documentation

10.4.3.1 workBackwardProblem()

```
void workBackwardProblem ( )
```

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

Definition at line 44 of file backwardproblem.cpp.

Here is the call graph for this function:

10.4.3.2 gett()

```
realtype gett ( ) const
```

accessor for t

Returns

t

Definition at line 32 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.3 getwhich()

```
int getwhich ( ) const
```

accessor for which

Returns

which

Definition at line 39 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.4 getwhichptr()

```
int* getwhichptr ( )
```

accessor for pointer to which

Returns

which

Definition at line 46 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.5 getxBptr()

```
AmiVector* getxBptr ( )
```

accessor for pointer to xB

Returns

&xB

Definition at line 53 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.6 getxQBptr()

```
AmiVector* getxQBptr ( )
```

accessor for pointer to xQB

Returns

&xQB

Definition at line 60 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.7 getdxBptr()

```
AmiVector* getdxBptr ( )
```

accessor for pointer to dxB

Returns

&dxB

Definition at line 67 of file backwardproblem.h.

Here is the caller graph for this function:

10.4.3.8 getdJydx()

```
std::vector<realtype> getdJydx ( ) const
```

accessor for dJydx

Returns

dJydx

Definition at line 74 of file backwardproblem.h.

Here is the caller graph for this function:

10.5 CvodeException Class Reference

cvode exception handler class

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

Inheritance diagram for CvodeException:

Public Member Functions

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

10.5.1 Detailed Description

Definition at line 117 of file exception.h.

10.5.2 Constructor & Destructor Documentation

10.5.2.1 CvodeException()

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

constructor

Parameters

in	*error_code*	error code returned by cvode function
in	*function*	cvode function name

Definition at line 123 of file exception.h.

10.6 ExpData Class Reference

ExpData carries all information about experimental or condition-specific data.

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

Public Member Functions

- ExpData ()
- ExpData (int nytrue, int nztrue, int nt, int nmaxevent)

  ExpData.
- ExpData (int nytrue, int nztrue, int nt, int nmaxevent, std::vector< realtype > const &my, std::vector< realtype > const &sigmay, std::vector< realtype > const &mz, std::vector< realtype > const &sigmaz)

  ExpData.
- ExpData (const Model &model)
- ExpData (const ExpData &other)

  Copy constructor.
- void setObservedData (const double *observedData)
- void setObservedDataStdDev (const double *observedDataStdDev)
- void setObservedEvents (const double *observedEvents)
- void setObservedEventsStdDev (const double *observedEventsStdDev)

Public Attributes

- std::vector< realtype > my
- std::vector< realtype > sigmay
- std::vector< realtype > mz
- std::vector< realtype > sigmaz
- const int nytrue
- const int nztrue
- const int nt
- const int nmaxevent
- std::vector< realtype > fixedParameters
- std::vector< realtype > fixedParametersPreequilibration

10.6.1 Detailed Description

Definition at line 13 of file edata.h.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 ExpData() [1/5]

```
ExpData ( )
```

default constructor

Definition at line 10 of file edata.cpp.

10.6.2.2 ExpData() [2/5]

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

Parameters

nytrue	
nztrue	
nt	
nmaxevent	

Definition at line 12 of file edata.cpp.

10.6.2.3 ExpData() [3/5]

```
ExpData (
            int nytrue,
            int nztrue,
            int nt,
            int nmaxevent,
            std::vector< realtype > const & my,
            std::vector< realtype > const & sigmay,
            std::vector< realtype > const & mz,
            std::vector< realtype > const & sigmaz )
```

Parameters

nytrue	
nztrue	
nt	
nmaxevent	
my	
sigmay	
mz	
sigmaz	

Definition at line 22 of file edata.cpp.

10.6.2.4 ExpData() [4/5]

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

constructor that initializes with Model

Parameters

model	pointer to model specification object **Type**: Model

Definition at line 33 of file edata.cpp.

10.6.2.5 ExpData() [5/5]

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

Parameters

other	object to copy from

Definition at line 38 of file edata.cpp.

10.6.3 Member Function Documentation

10.6.3.1 setObservedData()

```
void setObservedData (
            const double * observedData )
```

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

Parameters

observedData	observed data

Definition at line 44 of file edata.cpp.

Here is the caller graph for this function:

10.6.3.2 setObservedDataStdDev()

```
void setObservedDataStdDev (
            const double * observedDataStdDev )
```

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

Parameters

observedDataStdDev	standard deviation of observed data

Definition at line 55 of file edata.cpp.

Here is the caller graph for this function:

10.6.3.3 setObservedEvents()

```
void setObservedEvents (
            const double * observedEvents )
```

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

Parameters

observedEvents	observed event data

Definition at line 66 of file edata.cpp.

Here is the caller graph for this function:

10.6.3.4 setObservedEventsStdDev()

```
void setObservedEventsStdDev (
            const double * observedEventsStdDev )
```

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

Parameters

observedEventsStdDev	standard deviation of observed event data

Definition at line 77 of file edata.cpp.

Here is the caller graph for this function:

10.6.4 Member Data Documentation

10.6.4.1 my

```
std::vector<realtype> my
```

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

Definition at line 66 of file edata.h.

10.6.4.2 sigmay

```
std::vector<realtype> sigmay
```

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

Definition at line 68 of file edata.h.

10.6.4.3 mz

```
std::vector<realtype> mz
```

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

Definition at line 71 of file edata.h.

10.6.4.4 sigmaz

```
std::vector<realtype> sigmaz
```

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

Definition at line 74 of file edata.h.

10.6.4.5 nytrue

```
const int nytrue
```

number of observables

Definition at line 77 of file edata.h.

10.6.4.6 nztrue

```
const int nztrue
```

number of event observables

Definition at line 79 of file edata.h.

10.6.4.7 nt

```
const int nt
```

number of timepoints

Definition at line 81 of file edata.h.

10.6.4.8 nmaxevent

```
const int nmaxevent
```

maximal number of event occurences

Definition at line 83 of file edata.h.

10.6.4.9 fixedParameters

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

condition-specific parameters of size Model::nk() or empty

Definition at line 86 of file edata.h.

10.6.4.10 fixedParametersPreequilibration

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

condition-specific parameters for pre-equilibration of size Model::nk() or empty

Definition at line 88 of file edata.h.

10.7 ForwardProblem Class Reference

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

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

Public Member Functions

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

Public Attributes

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

10.7.1 Detailed Description

Definition at line 22 of file forwardproblem.h.

10.7.2 Constructor & Destructor Documentation

10.7.2.1 ForwardProblem()

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

default constructor

Parameters

rdata	pointer to ReturnData instance
edata	pointer to ExpData instance
model	pointer to Model instance
solver	pointer to Solver instance

Definition at line 40 of file forwardproblem.cpp.

Here is the call graph for this function:

10.7.2.2 ~ForwardProblem()

```
~ForwardProblem ( )
```

default destructor

Definition at line 27 of file forwardproblem.h.

10.7.3 Member Function Documentation

10.7.3.1 workForwardProblem()

```
void workForwardProblem ( )
```

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

Definition at line 75 of file forwardproblem.cpp.

Here is the call graph for this function:

10.7.3.2 getTime()

```
realtype getTime ( ) const
```

accessor for t

Returns

t

Definition at line 45 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.3.3 getStateSensitivity()

```
AmiVectorArray getStateSensitivity ( ) const
```

accessor for sx

Returns

sx

Definition at line 52 of file forwardproblem.h.

10.7.3.4 getStatesAtDiscontinuities()

```
AmiVectorArray getStatesAtDiscontinuities ( ) const
```

accessor for x_disc

Returns

x_disc

Definition at line 59 of file forwardproblem.h.

10.7.3.5 getRHSAtDiscontinuities()

```
AmiVectorArray getRHSAtDiscontinuities ( ) const
```

accessor for xdot_disc

Returns

xdot_disc

Definition at line 66 of file forwardproblem.h.

10.7.3.6 getRHSBeforeDiscontinuities()

```
AmiVectorArray getRHSBeforeDiscontinuities ( ) const
```

accessor for xdot_old_disc

Returns

xdot_old_disc

Definition at line 73 of file forwardproblem.h.

10.7.3.7 getNumberOfRoots()

```
std::vector<int> getNumberOfRoots ( ) const
```

accessor for nroots

Returns

nroots

Definition at line 80 of file forwardproblem.h.

10.7.3.8 getDiscontinuities()

```
std::vector<realtype> getDiscontinuities ( ) const
```

accessor for discs

Returns

discs

Definition at line 87 of file forwardproblem.h.

10.7.3.9 getRootIndexes()

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

accessor for rootidx

Returns

rootidx

Definition at line 94 of file forwardproblem.h.

10.7.3.10 getDJydx()

```
std::vector<realtype> getDJydx ( ) const
```

accessor for dJydx

Returns

dJydx

Definition at line 101 of file forwardproblem.h.

10.7.3.11 getDJzdx()

```
std::vector<realtype> getDJzdx ( ) const
```

accessor for dJzdx

Returns

dJzdx

Definition at line 108 of file forwardproblem.h.

10.7.3.12 getRootCounter()

```
int getRootCounter ( ) const
```

accessor for iroot

Returns

iroot

Definition at line 115 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.3.13 getStatePointer()

```
AmiVector* getStatePointer ( )
```

accessor for pointer to x

Returns

&x

Definition at line 122 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.3.14 getStateDerivativePointer()

```
AmiVector* getStateDerivativePointer ( )
```

accessor for pointer to dx

Returns

&dx

Definition at line 129 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.3.15 getStateSensitivityPointer()

```
AmiVectorArray* getStateSensitivityPointer ( )
```

accessor for pointer to sx

Returns

&sx

Definition at line 136 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.3.16 getStateDerivativeSensitivityPointer()

```
AmiVectorArray* getStateDerivativeSensitivityPointer ( )
```

accessor for pointer to sdx

Returns

&sdx

Definition at line 143 of file forwardproblem.h.

Here is the caller graph for this function:

10.7.4 Member Data Documentation

10.7.4.1 model

```
Model* model
```

pointer to model instance

Definition at line 34 of file forwardproblem.h.

10.7.4.2 rdata

```
ReturnData* rdata
```

pointer to return data instance

Definition at line 36 of file forwardproblem.h.

10.7.4.3 solver

```
Solver* solver
```

pointer to solver instance

Definition at line 38 of file forwardproblem.h.

10.7.4.4 edata

```
const ExpData* edata
```

pointer to experimental data instance

Definition at line 40 of file forwardproblem.h.

10.8 IDAException Class Reference

ida exception handler class

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

Inheritance diagram for IDAException:

Public Member Functions

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

10.8.1 Detailed Description

Definition at line 129 of file exception.h.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 IDAException()

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

constructor

Parameters

in	*error_code*	error code returned by ida function
in	*function*	ida function name

Definition at line 135 of file exception.h.

10.9 IntegrationFailure Class Reference

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

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

Inheritance diagram for IntegrationFailure:

Public Member Functions

- IntegrationFailure (int code, realtype t)

Public Attributes

- int error_code
- realtype time

10.9.1 Detailed Description

Definition at line 144 of file exception.h.

10.9.2 Constructor & Destructor Documentation

10.9.2.1 IntegrationFailure()

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

constructor

Parameters

in	*code*	error code returned by cvode/ida
in	*t*	time of integration failure

Definition at line 154 of file exception.h.

10.9.3 Member Data Documentation

10.9.3.1 error_code

```
int error_code
```

error code returned by cvode/ida

Definition at line 147 of file exception.h.

10.9.3.2 time

```
realtype time
```

time of integration failure

Definition at line 149 of file exception.h.

10.10 Model Class Reference

The Model class represents an AMICI ODE model. The model can compute various model related quantities based on symbolically generated code.

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

Inheritance diagram for Model:

Public Member Functions

- Model ()
- Model (const int nx, const int nxtrue, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const AMICI_o2mode o2mode, const std::vector< realtype > p, const std::vector< realtype > k, const std::vector< int > plist, const std::vector< realtype > idlist, const std::vector< int > z2event)
- Model & operator= (Model const &other)=delete
- Model (Model const &other)
- virtual Model * clone () const =0

  Clone this instance.
- void initializeVectors ()

  Set the nplist-dependent vectors to their proper sizes.
- virtual std::unique_ptr< Solver > getSolver ()=0
- virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root)=0
- virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot)=0
- virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J)=0
- virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J)=0
- virtual void fJDiag (realtype t, AmiVector *Jdiag, realtype cj, AmiVector *x, AmiVector *dx)=0
- virtual void fdxdotdp (realtype t, AmiVector *x, AmiVector *dx)=0
- virtual void fJv (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj)=0
- void fx0 (AmiVector *x)
- virtual void fdx0 (AmiVector *x0, AmiVector *dx0)
- void fsx0 (AmiVectorArray *sx, const AmiVector *x)
- virtual void fsdx0 ()
- void fstau (const realtype t, const int ie, const AmiVector *x, const AmiVectorArray *sx)
- void fy (int it, ReturnData *rdata)
- void fdydp (const int it, ReturnData *rdata)
- void fdydx (const int it, ReturnData *rdata)

- void fz (const int nroots, const int ie, const realtype t, const AmiVector ∗x, ReturnData ∗rdata)
- void fsz (const int nroots, const int ie, const realtype t, const AmiVector ∗x, const AmiVectorArray ∗sx, ReturnData ∗rdata)
- void frz (const int nroots, const int ie, const realtype t, const AmiVector ∗x, ReturnData ∗rdata)
- void fsrz (const int nroots, const int ie, const realtype t, const AmiVector ∗x, const AmiVectorArray ∗sx, ReturnData ∗rdata)
- void fdzdp (const realtype t, const int ie, const AmiVector ∗x)
- void fdzdx (const realtype t, const int ie, const AmiVector ∗x)
- void fdrzdp (const realtype t, const int ie, const AmiVector ∗x)
- void fdrzdx (const realtype t, const int ie, const AmiVector ∗x)
- void fdeltax (const int ie, const realtype t, const AmiVector ∗x, const AmiVector ∗xdot, const AmiVector ∗xdot↩_old)
- void fdeltasx (const int ie, const realtype t, const AmiVector ∗x, const AmiVectorArray ∗sx, const AmiVector ∗xdot, const AmiVector ∗xdot_old)
- void fdeltaxB (const int ie, const realtype t, const AmiVector ∗x, const AmiVector ∗xB, const AmiVector ∗xdot, const AmiVector ∗xdot_old)
- void fdeltaqB (const int ie, const realtype t, const AmiVector ∗x, const AmiVector ∗xB, const AmiVector ∗xdot, const AmiVector ∗xdot_old)
- void fsigma_y (const int it, const ExpData ∗edata, ReturnData ∗rdata)
- void fdsigma_ydp (const int it, const ReturnData ∗rdata)
- void fsigma_z (const realtype t, const int ie, const int ∗nroots, const ExpData ∗edata, ReturnData ∗rdata)
- void fdsigma_zdp (const realtype t)
- void fJy (const int it, ReturnData ∗rdata, const ExpData ∗edata)
- void fJz (const int nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fJrz (const int nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fdJydy (const int it, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJydsigma (const int it, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJzdz (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJzdsigma (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJrzdz (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJrzdsigma (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- virtual ∼Model ()
- void fsy (const int it, ReturnData ∗rdata)
- void fsz_tf (const int ∗nroots, const int ie, ReturnData ∗rdata)
- void fsJy (const int it, const std::vector< realtype > dJydx, ReturnData ∗rdata)
- void fdJydp (const int it, const ExpData ∗edata, const ReturnData ∗rdata)
- void fdJydx (std::vector< realtype > ∗dJydx, const int it, const ExpData ∗edata, const ReturnData ∗rdata)
- void fsJz (const int nroots, const std::vector< realtype > dJzdx, AmiVectorArray ∗sx, ReturnData ∗rdata)
- void fdJzdp (const int nroots, realtype t, const ExpData ∗edata, const ReturnData ∗rdata)
- void fdJzdx (std::vector< realtype > ∗dJzdx, const int nroots, realtype t, const ExpData ∗edata, const ReturnData ∗rdata)
- void initialize (AmiVector ∗x, AmiVector ∗dx)
- void initializeStates (AmiVector ∗x)
- void initHeaviside (AmiVector ∗x, AmiVector ∗dx)
- int nplist () const
- int np () const
- int nk () const
- const double ∗ k () const
- int nMaxEvent () const

  Get nmaxevent.
- void setNMaxEvent (int nmaxevent)

  setNMaxEvent
- int nt () const

  Get number of timepoints.

- std::vector< AMICI_parameter_scaling > getParameterScale () const

  getParameterScale
- void setParameterScale (AMICI_parameter_scaling pscale)

  setParameterScale
- void setParameterScale (std::vector< AMICI_parameter_scaling > pscale)

  setParameterScale
- std::vector< realtype > getParameters () const

  getParameters
- void setParameters (std::vector< realtype > const &p)

  setParameters
- std::vector< realtype > getUnscaledParameters () const

  getUnscaledParameters
- std::vector< realtype > getFixedParameters () const

  getFixedParameters
- void setFixedParameters (std::vector< realtype > const &k)

  setFixedParameters
- std::vector< realtype > getTimepoints () const

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

  setTimepoints
- double t (int idx) const

  Get timepoint for given index.
- std::vector< int > getParameterList () const

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

  setParameterList
- std::vector< realtype > getInitialStates () const

  getInitialStates
- void setInitialStates (std::vector< realtype > const &x0)

  setInitialStates
- std::vector< realtype > getInitialStateSensitivities () const

  getInitialStateSensitivities
- void setInitialStateSensitivities (std::vector< realtype > const &sx0)

  setInitialStateSensitivities
- double t0 () const
- void setT0 (double t0)

  setT0
- int plist (int pos) const
- void unscaleParameters (double *bufferUnscaled) const

  unscaleParameters
- void requireSensitivitiesForAllParameters ()

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

  Recurring terms in xdot.
- void fdwdp (const realtype t, const N_Vector x)

  Recurring terms in xdot, parameter derivative.
- void fdwdx (const realtype t, const N_Vector x)

  Recurring terms in xdot, state derivative.
- void updateHeaviside (const std::vector< int > rootsfound)
- void updateHeavisideB (const int *rootsfound)
- realtype gett (const int it, const ReturnData *rdata) const
- int checkFinite (const int N, const realtype *array, const char *fun) const

  Check if the given array has only finite elements. If not try to give hints by which other fields this could be caused.

Public Attributes

- const int nx
- const int nxtrue
- const int ny
- const int nytrue
- const int nz
- const int nztrue
- const int ne
- const int nw
- const int ndwdx
- const int ndwdp
- const int nnz
- const int nJ
- const int ubw
- const int lbw
- const AMICI_o2mode o2mode
- const std::vector< int > z2event
- const std::vector< realtype > idlist
- std::vector< realtype > sigmay
- std::vector< realtype > dsigmaydp
- std::vector< realtype > sigmaz
- std::vector< realtype > dsigmazdp
- std::vector< realtype > dJydp
- std::vector< realtype > dJzdp
- std::vector< realtype > deltax
- std::vector< realtype > deltasx
- std::vector< realtype > deltaxB
- std::vector< realtype > deltaqB
- std::vector< realtype > dxdotdp

Protected Member Functions

- virtual void fx0 (realtype *x0, const realtype t, const realtype *p, const realtype *k)
- virtual void fsx0 (realtype *sx0, const realtype t, const realtype *x0, const realtype *p, const realtype *k, const int ip)
- virtual void fstau (realtype *stau, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip, const int ie)
- virtual void fy (realtype *y, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdydp (realtype *dydp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void fdydx (realtype *dydx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fz (realtype *z, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fsz (realtype *sz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void frz (realtype *rz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fsrz (realtype *srz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void fdzdp (realtype *dzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)

- virtual void fdzdx (realtype *dzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdrzdp (realtype *drzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void fdrzdx (realtype *drzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdeltax (realtype *deltax, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old)
- virtual void fdeltasx (realtype *deltasx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *sx, const realtype *stau)
- virtual void fdeltaxB (realtype *deltaxB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void fdeltaqB (realtype *deltaqB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void fsigma_y (realtype *sigmay, const realtype t, const realtype *p, const realtype *k)
- virtual void fdsigma_ydp (realtype *dsigmaydp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void fsigma_z (realtype *sigmaz, const realtype t, const realtype *p, const realtype *k)
- virtual void fdsigma_zdp (realtype *dsigmazdp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void fJy (realtype *nllh, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fJz (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fJrz (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz)
- virtual void fdJydy (realtype *dJydy, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fdJydsigma (realtype *dJydsigma, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fdJzdz (realtype *dJzdz, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fdJzdsigma (realtype *dJzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fdJrzdz (realtype *dJrzdz, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)
- virtual void fdJrzdsigma (realtype *dJrzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)
- virtual void fw (realtype *w, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdwdp (realtype *dwdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)
- virtual void fdwdx (realtype *dwdx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)
- void getmy (const int it, const ExpData *edata)
- void getmz (const int nroots, const ExpData *edata)
- const realtype * gety (const int it, const ReturnData *rdata) const
- const realtype * getx (const int it, const ReturnData *rdata) const
- const realtype * getsx (const int it, const ReturnData *rdata) const
- const realtype * getz (const int nroots, const ReturnData *rdata) const
- const realtype * getrz (const int nroots, const ReturnData *rdata) const
- const realtype * getsz (const int nroots, const int nplist, const ReturnData *rdata) const
- const realtype * getsrz (const int nroots, const int nplist, const ReturnData *rdata) const

Protected Attributes

- SlsMat J = nullptr
- std::vector< realtype > my
- std::vector< realtype > mz
- std::vector< realtype > dJydy
- std::vector< realtype > dJydsigma
- std::vector< realtype > dJzdz
- std::vector< realtype > dJzdsigma
- std::vector< realtype > dJrzdz
- std::vector< realtype > dJrzdsigma
- std::vector< realtype > dzdx
- std::vector< realtype > dzdp
- std::vector< realtype > drzdx
- std::vector< realtype > drzdp
- std::vector< realtype > dydp
- std::vector< realtype > dydx
- std::vector< realtype > w
- std::vector< realtype > dwdx
- std::vector< realtype > dwdp
- std::vector< realtype > M
- std::vector< realtype > stau
- std::vector< realtype > h
- std::vector< realtype > unscaledParameters
- std::vector< realtype > originalParameters
- std::vector< realtype > fixedParameters
- std::vector< int > plist_
- std::vector< double > x0data
- std::vector< realtype > sx0data
- std::vector< realtype > ts
- int nmaxevent = 10
- std::vector< AMICI_parameter_scaling > pscale
- double tstart = 0.0

Friends

- template<class Archive >
  void boost::serialization::serialize (Archive &ar, Model &r, const unsigned int version)

  Serialize Model (see boost::serialization::serialize)
- bool operator== (const Model &a, const Model &b)

  Check equality of data members.

10.10.1 Detailed Description

Definition at line 41 of file model.h.

10.10.2 Constructor & Destructor Documentation

10.10.2.1 Model() [1/3]

```
Model ( )
```

default constructor

Definition at line 44 of file model.h.

10.10.2.2 Model() [2/3]

```
Model (
            const int nx,
            const int nxtrue,
            const int ny,
            const int nytrue,
            const int nz,
            const int nztrue,
            const int ne,
            const int nJ,
            const int nw,
            const int ndwdx,
            const int ndwdp,
            const int nnz,
            const int ubw,
            const int lbw,
            const AMICI_o2mode o2mode,
            const std::vector< realtype > p,
            const std::vector< realtype > k,
            const std::vector< int > plist,
            const std::vector< realtype > idlist,
            const std::vector< int > z2event )
```

constructor with model dimensions

Parameters

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

Definition at line 73 of file model.h.

10.10.2.3 Model() [3/3]

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

Copy constructor

Parameters

other	object to copy from

Returns

Definition at line 135 of file model.h.

10.10.2.4 ~Model()

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

default destructor

Definition at line 360 of file model.h.

10.10.3 Member Function Documentation

10.10.3.1 operator=()

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

Copy assignment is disabled until const members are removed

Parameters

other	object to copy from

Returns

10.10.3.2 clone()

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

Returns

The clone

10.10.3.3 getSolver()

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

Retrieves the solver object

Returns

The Solver instance

Implemented in Model_ODE, and Model_DAE.

10.10.3.4 froot()

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

Root function

Parameters

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

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.5 fxdot()

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

Residual function

Parameters

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

Implemented in Model_ODE, and Model_DAE.

10.10.3.6 fJ()

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

Dense Jacobian function

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xdot	values of residual function (unused)
J	dense matrix to which values of the jacobian will be written

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.7 fJSparse()

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

Sparse Jacobian function

Parameters

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

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.8 fJDiag()

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

Diagonal Jacobian function

Parameters

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

Returns

flag indicating successful evaluation

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.9 fdxdotdp()

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

parameter derivative of residual function

Parameters

t	time
x	state
dx	time derivative of state (DAE only)

Returns

flag indicating successful evaluation

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.10 fJv()

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

Jacobian multiply function

Parameters

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

Implemented in Model_ODE, and Model_DAE.

Here is the caller graph for this function:

10.10.3.11 fx0() [1/2]

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

Initial states

Parameters

x	pointer to state variables

Definition at line 320 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.12 fdx0()

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

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

Parameters

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

Definition at line 287 of file model.h.

Here is the caller graph for this function:

10.10.3.13 fsx0() [1/2]

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

Initial value for initial state sensitivities

Parameters

sx	pointer to state sensitivity variables
x	pointer to state variables

Definition at line 329 of file model.cpp.

Here is the call graph for this function:

10.10.3.14 fsdx0()

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

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

Definition at line 294 of file model.h.

10.10.3.15 fstau() [1/2]

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

Sensitivity of event timepoint, total derivative

Parameters

t	current timepoint
ie	event index
x	pointer to state variables
sx	pointer to state sensitivity variables

Definition at line 341 of file model.cpp.

Here is the call graph for this function:

10.10.3.16 fy() [1/2]

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

Observables / measurements

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 352 of file model.cpp.

Here is the call graph for this function:

10.10.3.17 fdydp() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 360 of file model.cpp.

Here is the call graph for this function:

10.10.3.18 fdydx() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 371 of file model.cpp.

Here is the call graph for this function:

10.10.3.19 fz() [1/2]

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

Event-resolved output

Parameters

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

Definition at line 383 of file model.cpp.

Here is the call graph for this function:

10.10.3.20 fsz() [1/2]

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

Sensitivity of z, total derivative

Parameters

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

Definition at line 395 of file model.cpp.

Here is the call graph for this function:

10.10.3.21 frz() [1/2]

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

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

Parameters

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

Definition at line 409 of file model.cpp.

Here is the call graph for this function:

10.10.3.22 fsrz() [1/2]

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

Sensitivity of rz, total derivative

Parameters

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

Definition at line 421 of file model.cpp.

Here is the call graph for this function:

10.10.3.23 fdzdp() [1/2]

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

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

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 432 of file model.cpp.

Here is the call graph for this function:

10.10.3.24 fdzdx() [1/2]

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

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

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 444 of file model.cpp.

Here is the call graph for this function:

10.10.3.25 fdrzdp() [1/2]

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

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

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 454 of file model.cpp.

Here is the call graph for this function:

10.10.3.26 fdrzdx() [1/2]

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

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

Parameters

ie	event index
t	current timepoint
x	current state

Definition at line 466 of file model.cpp.

Here is the call graph for this function:

10.10.3.27 fdeltax() [1/2]

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

State update functions for events

Parameters

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

Definition at line 478 of file model.cpp.

Here is the call graph for this function:

10.10.3.28 fdeltasx() [1/2]

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

Sensitivity update functions for events, total derivative

Parameters

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

Definition at line 492 of file model.cpp.

Here is the call graph for this function:

10.10.3.29 fdeltaxB() [1/2]

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

Adjoint state update functions for events

Parameters

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

Definition at line 509 of file model.cpp.

Here is the call graph for this function:

10.10.3.30 fdeltaqB() [1/2]

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

Quadrature state update functions for events

Parameters

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

Definition at line 523 of file model.cpp.

Here is the call graph for this function:

10.10.3.31 fsigma_y() [1/2]

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

Standard deviation of measurements

Parameters

it	timepoint index
edata	pointer to experimental data instance
rdata	pointer to return data instance

Definition at line 536 of file model.cpp.

Here is the call graph for this function:

10.10.3.32 fdsigma_ydp() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 556 of file model.cpp.

10.10.3.33 fsigma_z() [1/2]

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

Standard deviation of events

Parameters

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

Definition at line 569 of file model.cpp.

Here is the call graph for this function:

10.10.3.34 fdsigma_zdp() [1/2]

```
void fdsigma_zdp (
            const realtype t )
```

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

Parameters

t	current timepoint

Definition at line 588 of file model.cpp.

Here is the call graph for this function:

10.10.3.35 fJy() [1/2]

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

negative log-likelihood of measurements y

Parameters

it	timepoint index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 599 of file model.cpp.

Here is the call graph for this function:

10.10.3.36 fJz() [1/2]

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

negative log-likelihood of event-resolved measurements z

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 616 of file model.cpp.

Here is the call graph for this function:

10.10.3.37 fJrz() [1/2]

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

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

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 634 of file model.cpp.

Here is the call graph for this function:

10.10.3.38 fdJydy() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 651 of file model.cpp.

Here is the call graph for this function:

10.10.3.39 fdJydsigma() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 668 of file model.cpp.

Here is the call graph for this function:

10.10.3.40 fdJzdz() [1/2]

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

partial derivative of event measurement negative log-likelihood Jz

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 684 of file model.cpp.

Here is the call graph for this function:

10.10.3.41 fdJzdsigma() [1/2]

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

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

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 701 of file model.cpp.

Here is the call graph for this function:

10.10.3.42 fdJrzdz() [1/2]

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

partial derivative of event measurement negative log-likelihood Jz

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 717 of file model.cpp.

Here is the call graph for this function:

10.10.3.43 fdJrzdsigma() [1/2]

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

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

Parameters

nroots	event index
rdata	pointer to return data instance
edata	pointer to experimental data instance

Definition at line 734 of file model.cpp.

Here is the call graph for this function:

10.10.3.44 fsy()

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

Sensitivity of measurements y, total derivative

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 15 of file model.cpp.

Here is the call graph for this function:

10.10.3.45 fsz_tf()

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

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

Parameters

nroots	number of events for event index
ie	event index
rdata	pointer to return data instance

Definition at line 38 of file model.cpp.

Here is the call graph for this function:

10.10.3.46 fsJy()

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

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

Parameters

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

Definition at line 52 of file model.cpp.

Here is the call graph for this function:

10.10.3.47 fdJydp()

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

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

Parameters

it	timepoint index
edata	pointer to experimental data instance
rdata	pointer to return data instance

Definition at line 87 of file model.cpp.

Here is the call graph for this function:

10.10.3.48 fdJydx()

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

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

Parameters

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

Definition at line 116 of file model.cpp.

Here is the call graph for this function:

10.10.3.49 fsJz()

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

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

Parameters

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

Definition at line 142 of file model.cpp.

Here is the call graph for this function:

10.10.3.50 fdJzdp()

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

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

Parameters

nroots	event index
t	current timepoint
edata	pointer to experimental data instance
rdata	pointer to return data instance

Definition at line 183 of file model.cpp.

Here is the call graph for this function:

10.10.3.51 fdJzdx()

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

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

Parameters

dJzdx	pointer to vector with values of state derivative of Jz
nroots	event index
t	current timepoint
edata	pointer to experimental data instance
rdata	pointer to return data instance

Definition at line 227 of file model.cpp.

Here is the call graph for this function:

10.10.3.52 initialize()

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

initialization of model properties

Parameters

x	pointer to state variables
dx	pointer to time derivative of states (DAE only)

Definition at line 254 of file model.cpp.

Here is the call graph for this function:

10.10.3.53 initializeStates()

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

initialization of initial states

Parameters

x	pointer to state variables

Definition at line 267 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.54 initHeaviside()

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

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

Parameters

x	pointer to state variables
dx	pointer to time derivative of states (DAE only)

Definition at line 284 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.55 nplist()

```
int nplist ( ) const
```

number of paramaeters wrt to which sensitivities are computed

Returns

length of sensitivity index vector

Definition at line 392 of file model.h.

Here is the caller graph for this function:

10.10.3.56 np()

```
int np ( ) const
```

total number of model parameters

Returns

length of parameter vector

Definition at line 398 of file model.h.

Here is the caller graph for this function:

10.10.3.57 nk()

```
int nk ( ) const
```

number of constants

Returns

length of constant vector

Definition at line 404 of file model.h.

Here is the caller graph for this function:

10.10.3.58 k()

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

constants

Returns

pointer to constants array

Definition at line 411 of file model.h.

Here is the caller graph for this function:

10.10.3.59 nMaxEvent()

```
int nMaxEvent ( ) const
```

Returns

nmaxevent

Definition at line 419 of file model.h.

10.10.3.60 setNMaxEvent()

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

Parameters

nmaxevent	

Definition at line 427 of file model.h.

10.10.3.61 nt()

```
int nt ( ) const
```

Returns

number of timepoints

Definition at line 435 of file model.h.

10.10.3.62 getParameterScale()

```
std::vector<AMICI_parameter_scaling> getParameterScale ( ) const
```

Returns

Definition at line 443 of file model.h.

10.10.3.63 setParameterScale() [1/2]

```
void setParameterScale (
            AMICI_parameter_scaling pscale )
```

Parameters

pscale	

Definition at line 451 of file model.h.

Here is the call graph for this function:

10.10.3.64 setParameterScale() [2/2]

```
void setParameterScale (
            std::vector< AMICI_parameter_scaling > pscale )
```

Parameters

pscale	

Definition at line 461 of file model.h.

Here is the call graph for this function:

10.10.3.65 getParameters()

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

Returns

The user-set parameters (see also getUnscaledParameters)

Definition at line 471 of file model.h.

10.10.3.66 setParameters()

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

Parameters

p	

Definition at line 479 of file model.h.

Here is the call graph for this function:

10.10.3.67 getUnscaledParameters()

```
std::vector<realtype> getUnscaledParameters ( ) const
```

Returns

The unscaled parameters

Definition at line 491 of file model.h.

10.10.3.68 getFixedParameters()

```
std::vector<realtype> getFixedParameters ( ) const
```

Returns

Definition at line 499 of file model.h.

Here is the caller graph for this function:

10.10.3.69 setFixedParameters()

```
void setFixedParameters (
            std::vector< realtype > const & k )
```

Parameters

k	

Definition at line 507 of file model.h.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.70 getTimepoints()

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

Returns

Definition at line 517 of file model.h.

10.10.3.71 setTimepoints()

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

Parameters

ts	

Definition at line 525 of file model.h.

10.10.3.72 t()

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

Parameters

idx	

Returns

Definition at line 534 of file model.h.

Here is the caller graph for this function:

10.10.3.73 getParameterList()

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

Returns

Definition at line 542 of file model.h.

10.10.3.74 setParameterList()

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

Parameters

plist	

Definition at line 550 of file model.h.

Here is the call graph for this function:

10.10.3.75 getInitialStates()

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

Returns

Definition at line 563 of file model.h.

10.10.3.76 setInitialStates()

```
void setInitialStates (
            std::vector< realtype > const & x0 )
```

Parameters

x0	

Definition at line 571 of file model.h.

10.10.3.77 getInitialStateSensitivities()

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

Returns

Definition at line 581 of file model.h.

10.10.3.78 setInitialStateSensitivities()

```
void setInitialStateSensitivities (
            std::vector< realtype > const & sx0 )
```

Parameters

sx0	

Definition at line 589 of file model.h.

Here is the call graph for this function:

10.10.3.79 t0()

```
double t0 ( ) const
```

initial timepoint

Returns

timepoint

Definition at line 598 of file model.h.

Here is the caller graph for this function:

10.10.3.80 setT0()

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

Parameters

t0	

Definition at line 606 of file model.h.

Here is the call graph for this function:

10.10.3.81 plist()

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

entry in parameter list

Parameters

pos	index

Returns

entry

Definition at line 614 of file model.h.

Here is the caller graph for this function:

10.10.3.82 unscaleParameters()

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

Parameters

bufferUnscaled	

unscaleParameters removes parameter scaling according to the parameter scaling in pscale

Parameters

out	*bufferUnscaled*	unscaled parameters are written to the array **Type**: double

Returns

status flag indicating success of execution
Type: int

Definition at line 897 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.83 fw() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 749 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.84 fdwdp() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 759 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.85 fdwdx() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 770 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.86 updateHeaviside()

```
void updateHeaviside (
            const std::vector< int > rootsfound )
```

updateHeaviside updates the heaviside variables h on event occurences

Parameters

rootsfound	provides the direction of the zero-crossing, so adding it will give the right update to the heaviside variables (zero if no root was found)

Definition at line 708 of file model.h.

10.10.3.87 updateHeavisideB()

```
void updateHeavisideB (
            const int * rootsfound )
```

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

Parameters

rootsfound	provides the direction of the zero-crossing, so adding it will give the right update to the heaviside variables (zero if no root was found)

Definition at line 722 of file model.h.

10.10.3.88 gett()

```
realtype gett (
            const int it,
            const ReturnData * rdata ) const
```

get current timepoint from index

Parameters

it	timepoint index
rdata	pointer to return data instance

Returns

current timepoint

Definition at line 825 of file model.cpp.

10.10.3.89 checkFinite()

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

Parameters

N	
array	
fun	

Returns

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

Definition at line 883 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.90 fx0() [2/2]

```
virtual void fx0 (
            realtype * x0,
            const realtype t,
            const realtype * p,
            const realtype * k )  [protected], [virtual]
```

model specific implementation of fx0

Parameters

x0	initial state
t	initial time
p	parameter vector
k	constant vector

Definition at line 765 of file model.h.

10.10.3.91 fsx0() [2/2]

```
virtual void fsx0 (
            realtype * sx0,
            const realtype t,
            const realtype * x0,
            const realtype * p,
            const realtype * k,
            const int ip )  [protected], [virtual]
```

model specific implementation of fsx0

Parameters

sx0	initial state sensitivities
t	initial time
x0	initial state
p	parameter vector
k	constant vector
ip	sensitivity index

Definition at line 777 of file model.h.

10.10.3.92 fstau() [2/2]

```
virtual void fstau (
            realtype * stau,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * sx,
            const int ip,
            const int ie )  [protected], [virtual]
```

model specific implementation of fstau

Parameters

stau	total derivative of event timepoint
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
sx	current state sensitivity
ip	sensitivity index
ie	event index

Definition at line 792 of file model.h.

10.10.3.93 fy() [2/2]

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

model specific implementation of fy

Parameters

y	model output at current timepoint
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 804 of file model.h.

10.10.3.94 fdydp() [2/2]

```
virtual void fdydp (
            realtype * dydp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [protected], [virtual]
```

model specific implementation of fdydp

Parameters

dydp	partial derivative of observables y w.r.t. model parameters p
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ip	parameter index w.r.t. which the derivative is requested

Definition at line 817 of file model.h.

10.10.3.95 fdydx() [2/2]

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

model specific implementation of fdydx

Parameters

dydx	partial derivative of observables y w.r.t. model states x
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 829 of file model.h.

10.10.3.96 fz() [2/2]

```
virtual void fz (
            realtype * z,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation of fz

Parameters

z	value of event output
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 842 of file model.h.

10.10.3.97 fsz() [2/2]

```
virtual void fsz (
            realtype * sz,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * sx,
            const int ip )  [protected], [virtual]
```

model specific implementation of fsz

Parameters

sz	Sensitivity of rz, total derivative
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
sx	current state sensitivity
ip	sensitivity index

Definition at line 857 of file model.h.

10.10.3.98 frz() [2/2]

```
virtual void frz (
            realtype * rz,
            const int ie,
            const realtype t,
            const realtype * x,
```

```
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation of frz

Parameters

rz	value of root function at current timepoint (non-output events not included)
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 870 of file model.h.

10.10.3.99 fsrz() [2/2]

```
virtual void fsrz (
            realtype * srz,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * sx,
            const int ip )  [protected], [virtual]
```

model specific implementation of fsrz

Parameters

srz	Sensitivity of rz, total derivative
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
sx	current state sensitivity
h	heavyside vector
ip	sensitivity index

Definition at line 885 of file model.h.

10.10.3.100 fdzdp() [2/2]

```
virtual void fdzdp (
            realtype * dzdp,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [protected], [virtual]
```

model specific implementation of fdzdp

Parameters

dzdp	partial derivative of event-resolved output z w.r.t. model parameters p
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ip	parameter index w.r.t. which the derivative is requested

Definition at line 899 of file model.h.

10.10.3.101 fdzdx() [2/2]

```
virtual void fdzdx (
            realtype * dzdx,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation of fdzdx

Parameters

dzdx	partial derivative of event-resolved output z w.r.t. model states x
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 912 of file model.h.

10.10.3.102 fdrzdp() [2/2]

```
virtual void fdrzdp (
            realtype * drzdp,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [protected], [virtual]
```

model specific implementation of fdrzdp

Parameters

drzdp	partial derivative of root output rz w.r.t. model parameters p
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ip	parameter index w.r.t. which the derivative is requested

Definition at line 926 of file model.h.

10.10.3.103 fdrzdx() [2/2]

```
virtual void fdrzdx (
            realtype * drzdx,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation of fdrzdx

Parameters

drzdx	partial derivative of root output rz w.r.t. model states x
ie	event index
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector

Definition at line 939 of file model.h.

10.10.3.104 fdeltax() [2/2]

```
virtual void fdeltax (
            realtype * deltax,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ie,
            const realtype * xdot,
            const realtype * xdot_old )  [protected], [virtual]
```

model specific implementation of fdeltax

Parameters

deltax	state update
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ie	event index
xdot	new model right hand side
xdot_old	previous model right hand side

Definition at line 954 of file model.h.

10.10.3.105 fdeltasx() [2/2]

```
virtual void fdeltasx (
            realtype * deltasx,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w,
            const int ip,
            const int ie,
            const realtype * xdot,
            const realtype * xdot_old,
            const realtype * sx,
            const realtype * stau )  [protected], [virtual]
```

model specific implementation of fdeltasx

Parameters

deltasx	sensitivity update
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
w	repeating elements vector
ip	sensitivity index
ie	event index
xdot	new model right hand side
xdot_old	previous model right hand side
sx	state sensitivity
stau	event-time sensitivity

Definition at line 974 of file model.h.

10.10.3.106 fdeltaxB() [2/2]

```
virtual void fdeltaxB (
            realtype * deltaxB,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ie,
            const realtype * xdot,
            const realtype * xdot_old,
            const realtype * xB )  [protected], [virtual]
```

model specific implementation of fdeltaxB

Parameters

deltaxB	adjoint state update
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ie	event index
xdot	new model right hand side
xdot_old	previous model right hand side
xB	current adjoint state

Definition at line 992 of file model.h.

10.10.3.107 fdeltaqB() [2/2]

```
virtual void fdeltaqB (
            realtype * deltaqB,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip,
            const int ie,
            const realtype * xdot,
            const realtype * xdot_old,
            const realtype * xB )  [protected], [virtual]
```

model specific implementation of fdeltaqB

Parameters

deltaqB	sensitivity update
t	current time
x	current state
p	parameter vector
k	constant vector
h	heavyside vector
ip	sensitivity index
ie	event index
xdot	new model right hand side
xdot_old	previous model right hand side
xB	adjoint state

Definition at line 1010 of file model.h.

10.10.3.108 fsigma_y() [2/2]

```
virtual void fsigma_y (
            realtype * sigmay,
            const realtype t,
            const realtype * p,
            const realtype * k )  [protected], [virtual]
```

model specific implementation of fsigmay

Parameters

sigmay	standard deviation of measurements
t	current time
p	parameter vector
k	constant vector

Definition at line 1021 of file model.h.

10.10.3.109 fdsigma_ydp() [2/2]

```
virtual void fdsigma_ydp (
            realtype * dsigmaydp,
            const realtype t,
            const realtype * p,
            const realtype * k,
            const int ip )  [protected], [virtual]
```

model specific implementation of fsigmay

Parameters

dsigmaydp	partial derivative of standard deviation of measurements
t	current time
p	parameter vector
k	constant vector
ip	sensitivity index

Definition at line 1032 of file model.h.

10.10.3.110 fsigma_z() [2/2]

```
virtual void fsigma_z (
            realtype * sigmaz,
            const realtype t,
            const realtype * p,
            const realtype * k )  [protected], [virtual]
```

model specific implementation of fsigmaz

Parameters

sigmaz	standard deviation of event measurements
t	current time
p	parameter vector
k	constant vector

Definition at line 1042 of file model.h.

10.10.3.111 fdsigma_zdp() [2/2]

```
virtual void fdsigma_zdp (
            realtype * dsigmazdp,
            const realtype t,
            const realtype * p,
```

```
            const realtype * k,
            const int ip )  [protected], [virtual]
```

model specific implementation of fsigmaz

Parameters

dsigmazdp	partial derivative of standard deviation of event measurements
t	current time
p	parameter vector
k	constant vector
ip	sensitivity index

Definition at line 1053 of file model.h.

10.10.3.112 fJy() [2/2]

```
virtual void fJy (
            realtype * nllh,
            const int iy,
            const realtype * p,
            const realtype * k,
            const realtype * y,
            const realtype * sigmay,
            const realtype * my )  [protected], [virtual]
```

model specific implementation of fJy

Parameters

nllh	negative log-likelihood for measurements y
iy	output index
p	parameter vector
k	constant vector
y	model output at timepoint
sigmay	measurement standard deviation at timepoint
my	measurements at timepoint

Definition at line 1066 of file model.h.

10.10.3.113 fJz() [2/2]

```
virtual void fJz (
            realtype * nllh,
            const int iz,
            const realtype * p,
            const realtype * k,
            const realtype * z,
```

```
            const realtype * sigmaz,
            const realtype * mz )  [protected], [virtual]
```

model specific implementation of fJz

Parameters

nllh	negative log-likelihood for event measurements z
iz	event output index
p	parameter vector
k	constant vector
z	model event output at timepoint
sigmaz	event measurement standard deviation at timepoint
mz	event measurements at timepoint

Definition at line 1078 of file model.h.

10.10.3.114 fJrz() [2/2]

```
virtual void fJrz (
            realtype * nllh,
            const int iz,
            const realtype * p,
            const realtype * k,
            const realtype * z,
            const realtype * sigmaz )  [protected], [virtual]
```

model specific implementation of fJrz

Parameters

nllh	regularization for event measurements z
iz	event output index
p	parameter vector
k	constant vector
z	model event output at timepoint
sigmaz	event measurement standard deviation at timepoint

Definition at line 1090 of file model.h.

10.10.3.115 fdJydy() [2/2]

```
virtual void fdJydy (
            realtype * dJydy,
            const int iy,
            const realtype * p,
            const realtype * k,
            const realtype * y,
```

```
            const realtype * sigmay,
            const realtype * my )  [protected], [virtual]
```

model specific implementation of fdJydy

Parameters

dJydy	partial derivative of time-resolved measurement negative log-likelihood Jy
iy	output index
p	parameter vector
k	constant vector
y	model output at timepoint
sigmay	measurement standard deviation at timepoint
my	measurement at timepoint

Definition at line 1103 of file model.h.

10.10.3.116 fdJydsigma() [2/2]

```
virtual void fdJydsigma (
            realtype * dJydsigma,
            const int iy,
            const realtype * p,
            const realtype * k,
            const realtype * y,
            const realtype * sigmay,
            const realtype * my )  [protected], [virtual]
```

model specific implementation of fdJydsigma

Parameters

dJydsigma	Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigmay
iy	output index
p	parameter vector
k	constant vector
y	model output at timepoint
sigmay	measurement standard deviation at timepoint
my	measurement at timepoint

Definition at line 1118 of file model.h.

10.10.3.117 fdJzdz() [2/2]

```
virtual void fdJzdz (
            realtype * dJzdz,
            const int iz,
            const realtype * p,
```

```
            const realtype * k,
            const realtype * z,
            const realtype * sigmaz,
            const realtype * mz )  [protected], [virtual]
```

model specific implementation of fdJzdz

Parameters

dJzdz	partial derivative of event measurement negative log-likelihood Jz
iz	event output index
p	parameter vector
k	constant vector
z	model event output at timepoint
sigmaz	event measurement standard deviation at timepoint
mz	event measurement at timepoint

Definition at line 1132 of file model.h.

10.10.3.118 fdJzdsigma() [2/2]

```
virtual void fdJzdsigma (
            realtype * dJzdsigma,
            const int iz,
            const realtype * p,
            const realtype * k,
            const realtype * z,
            const realtype * sigmaz,
            const realtype * mz )  [protected], [virtual]
```

model specific implementation of fdJzdsigma

Parameters

dJzdsigma	Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz
iz	event output index
p	parameter vector
k	constant vector
z	model event output at timepoint
sigmaz	event measurement standard deviation at timepoint
mz	event measurement at timepoint

Definition at line 1147 of file model.h.

10.10.3.119 fdJrzdz() [2/2]

```
virtual void fdJrzdz (
            realtype * dJrzdz,
```

```
            const int iz,
            const realtype * p,
            const realtype * k,
            const realtype * rz,
            const realtype * sigmaz )  [protected], [virtual]
```

model specific implementation of fdJrzdz

Parameters

dJrzdz	partial derivative of event penalization Jrz
iz	event output index
p	parameter vector
k	constant vector
rz	model root output at timepoint
sigmaz	event measurement standard deviation at timepoint

Definition at line 1160 of file model.h.

10.10.3.120 fdJrzdsigma() [2/2]

```
virtual void fdJrzdsigma (
            realtype * dJrzdsigma,
            const int iz,
            const realtype * p,
            const realtype * k,
            const realtype * rz,
            const realtype * sigmaz )  [protected], [virtual]
```

model specific implementation of fdJrzdsigma

Parameters

dJrzdsigma	Sensitivity of event penalization Jrz w.r.t. standard deviation sigmaz
iz	event output index
p	parameter vector
k	constant vector
rz	model root output at timepoint
sigmaz	event measurement standard deviation at timepoint

Definition at line 1174 of file model.h.

10.10.3.121 fw() [2/2]

```
virtual void fw (
            realtype * w,
            const realtype t,
            const realtype * x,
```

```
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation of fw

Parameters

w	Recurring terms in xdot
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector

Definition at line 1187 of file model.h.

10.10.3.122 fdwdp() [2/2]

```
virtual void fdwdp (
            realtype * dwdp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w )  [protected], [virtual]
```

model specific implementation of dwdp

Parameters

dwdp	Recurring terms in xdot, parameter derivative
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

Definition at line 1199 of file model.h.

10.10.3.123 fdwdx() [2/2]

```
virtual void fdwdx (
            realtype * dwdx,
            const realtype t,
            const realtype * x,
```

```
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w )  [protected], [virtual]
```

model specific implementation of dwdx

Parameters

dwdx	Recurring terms in xdot, state derivative
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

Definition at line 1211 of file model.h.

10.10.3.124 getmy()

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

create my slice at timepoint

Parameters

it	timepoint index
edata	pointer to experimental data instance

Definition at line 780 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.125 getmz()

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

create mz slice at event

Parameters

nroots	event occurence
edata	pointer to experimental data instance

Definition at line 833 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.126 gety()

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

create y slice at timepoint

Parameters

it	timepoint index
rdata	pointer to return data instance

Returns

y y-slice from rdata instance

Definition at line 815 of file model.cpp.

Here is the caller graph for this function:

10.10.3.127 getx()

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

create x slice at timepoint

Parameters

it	timepoint index
rdata	pointer to return data instance

Returns

x x-slice from rdata instance

Definition at line 797 of file model.cpp.

Here is the caller graph for this function:

10.10.3.128 getsx()

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

create sx slice at timepoint

Parameters

it	timepoint index
rdata	pointer to return data instance

Returns

sx sx-slice from rdata instance

Definition at line 806 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.10.3.129 getz()

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

create z slice at event

Parameters

nroots	event occurence
rdata	pointer to return data instance

Returns

z slice

Definition at line 850 of file model.cpp.

Here is the caller graph for this function:

10.10.3.130 getrz()

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

create rz slice at event

Parameters

nroots	event occurence
rdata	pointer to return data instance

Returns

rz slice

Definition at line 859 of file model.cpp.

Here is the caller graph for this function:

10.10.3.131 getsz()

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

create sz slice at event

Parameters

nroots	event occurence
ip	sensitivity index
rdata	pointer to return data instance

Returns

z slice

Definition at line 869 of file model.cpp.

Here is the call graph for this function:

10.10.3.132 getsrz()

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

create srz slice at event

Parameters

nroots	event occurence
ip	sensitivity index
rdata	pointer to return data instance

Returns

rz slice

Definition at line 879 of file model.cpp.

Here is the call graph for this function:

10.10.4 Friends And Related Function Documentation

10.10.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
            Archive & ar,
            Model & r,
            const unsigned int version )  [friend]
```

Parameters

ar	Archive to serialize to
r	Data to serialize
version	Version number

10.10.4.2 operator==

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

Parameters

a	
b	

Returns

Definition at line 923 of file model.cpp.

10.10.5 Member Data Documentation

10.10.5.1 nx

```
const int nx
```

number of states

Definition at line 635 of file model.h.

10.10.5.2 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 637 of file model.h.

10.10.5.3 ny

```
const int ny
```

number of observables

Definition at line 639 of file model.h.

10.10.5.4 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 641 of file model.h.

10.10.5.5 nz

```
const int nz
```

number of event outputs

Definition at line 643 of file model.h.

10.10.5.6 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 645 of file model.h.

10.10.5.7 ne

```
const int ne
```

number of events

Definition at line 647 of file model.h.

10.10.5.8 nw

```
const int nw
```

number of common expressions

Definition at line 649 of file model.h.

10.10.5.9 ndwdx

```
const int ndwdx
```

number of derivatives of common expressions wrt x

Definition at line 651 of file model.h.

10.10.5.10 ndwdp

```
const int ndwdp
```

number of derivatives of common expressions wrt p

Definition at line 653 of file model.h.

10.10.5.11 nnz

```
const int nnz
```

number of nonzero entries in jacobian

Definition at line 655 of file model.h.

10.10.5.12 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 657 of file model.h.

10.10.5.13 ubw

```
const int ubw
```

upper bandwith of the jacobian

Definition at line 659 of file model.h.

10.10.5.14 lbw

```
const int lbw
```

lower bandwith of the jacobian

Definition at line 661 of file model.h.

10.10.5.15 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 664 of file model.h.

10.10.5.16 z2event

```
const std::vector<int> z2event
```

index indicating to which event an event output belongs

Definition at line 666 of file model.h.

10.10.5.17 idlist

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

flag array for DAE equations

Definition at line 668 of file model.h.

10.10.5.18 sigmay

```
std::vector<realtype> sigmay
```

data standard deviation

Definition at line 671 of file model.h.

10.10.5.19 dsigmaydp

```
std::vector<realtype> dsigmaydp
```

parameter derivative of data standard deviation

Definition at line 673 of file model.h.

10.10.5.20 sigmaz

```
std::vector<realtype> sigmaz
```

event standard deviation

Definition at line 675 of file model.h.

10.10.5.21 dsigmazdp

```
std::vector<realtype> dsigmazdp
```

parameter derivative of event standard deviation

Definition at line 677 of file model.h.

10.10.5.22 dJydp

```
std::vector<realtype> dJydp
```

parameter derivative of data likelihood

Definition at line 679 of file model.h.

10.10.5.23 dJzdp

```
std::vector<realtype> dJzdp
```

parameter derivative of event likelihood

Definition at line 681 of file model.h.

10.10.5.24 deltax

```
std::vector<realtype> deltax
```

change in x

Definition at line 684 of file model.h.

10.10.5.25 deltasx

```
std::vector<realtype> deltasx
```

change in sx

Definition at line 686 of file model.h.

10.10.5.26 deltaxB

```
std::vector<realtype> deltaxB
```

change in xB

Definition at line 688 of file model.h.

10.10.5.27 deltaqB

```
std::vector<realtype> deltaqB
```

change in qB

Definition at line 690 of file model.h.

10.10.5.28 dxdotdp

```
std::vector<realtype> dxdotdp
```

tempory storage of dxdotdp data across functions

Definition at line 693 of file model.h.

10.10.5.29 J

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

Jacobian

Definition at line 1233 of file model.h.

10.10.5.30 my

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

current observable

Definition at line 1236 of file model.h.

10.10.5.31 mz

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

current event measurement

Definition at line 1238 of file model.h.

10.10.5.32 dJydy

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

observable derivative of data likelihood

Definition at line 1240 of file model.h.

10.10.5.33 dJydsigma

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

observable sigma derivative of data likelihood

Definition at line 1242 of file model.h.

10.10.5.34 dJzdz

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

event ouput derivative of event likelihood

Definition at line 1244 of file model.h.

10.10.5.35 dJzdsigma

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

event sigma derivative of event likelihood

Definition at line 1246 of file model.h.

10.10.5.36 dJrzdz

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

event ouput derivative of event likelihood at final timepoint

Definition at line 1248 of file model.h.

10.10.5.37 dJrzdsigma

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

event sigma derivative of event likelihood at final timepoint

Definition at line 1250 of file model.h.

10.10.5.38 dzdx

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

state derivative of event output

Definition at line 1252 of file model.h.

10.10.5.39 dzdp

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

parameter derivative of event output

Definition at line 1254 of file model.h.

10.10.5.40 drzdx

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

state derivative of event timepoint

Definition at line 1256 of file model.h.

10.10.5.41 drzdp

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

parameter derivative of event timepoint

Definition at line 1258 of file model.h.

10.10.5.42 dydp

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

parameter derivative of observable

Definition at line 1260 of file model.h.

10.10.5.43 dydx

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

state derivative of observable

Definition at line 1262 of file model.h.

10.10.5.44 w

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

tempory storage of w data across functions

Definition at line 1264 of file model.h.

10.10.5.45 dwdx

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

tempory storage of dwdx data across functions

Definition at line 1266 of file model.h.

10.10.5.46 dwdp

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

tempory storage of dwdp data across functions

Definition at line 1268 of file model.h.

10.10.5.47 M

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

tempory storage of M data across functions

Definition at line 1270 of file model.h.

10.10.5.48 stau

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

tempory storage of stau data across functions

Definition at line 1272 of file model.h.

10.10.5.49 h

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

flag indicating whether a certain heaviside function should be active or not

Definition at line 1276 of file model.h.

10.10.5.50 unscaledParameters

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

unscaled parameters

Definition at line 1279 of file model.h.

10.10.5.51 originalParameters

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

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

Definition at line 1282 of file model.h.

10.10.5.52 fixedParameters

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

constants

Definition at line 1285 of file model.h.

10.10.5.53 plist_

```
std::vector<int> plist_ [protected]
```

indexes of parameters wrt to which sensitivities are computed

Definition at line 1288 of file model.h.

10.10.5.54 x0data

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

state initialisation (size nx)

Definition at line 1291 of file model.h.

10.10.5.55 sx0data

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

sensitivity initialisation (size nx $*$ nplist)

Definition at line 1294 of file model.h.

10.10.5.56 ts

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

timepoints (size nt)

Definition at line 1297 of file model.h.

10.10.5.57 nmaxevent

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

maximal number of events to track

Definition at line 1300 of file model.h.

10.10.5.58 pscale

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

parameter transformation of p

Definition at line 1303 of file model.h.

10.10.5.59 tstart

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

starting time

Definition at line 1306 of file model.h.

10.11 Model_DAE Class Reference

The Model class represents an AMICI DAE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchroneously.

```
#include <model_dae.h>
```

Inheritance diagram for Model_DAE:

Public Member Functions

- Model_DAE ()
- Model_DAE (const int nx, const int nxtrue, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const AMICI_o2mode o2mode, const std::vector< realtype > p, const std::vector< realtype > k, const std::vector< int > plist, const std::vector< realtype > idlist, const std::vector< int > z2event)
- virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J) override
- void fJ (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xdot, DlsMat J)
- void fJB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, DlsMat JB)
- virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J) override
- void fJSparse (realtype t, realtype cj, N_Vector x, N_Vector dx, SlsMat J)
- void fJSparseB (realtype t, realtype cj, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, SlsMat JB)
- virtual void fJDiag (realtype t, AmiVector *JDiag, realtype cj, AmiVector *x, AmiVector *dx) override
- virtual void fJv (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj) override
- void fJv (realtype t, N_Vector x, N_Vector dx, N_Vector v, N_Vector Jv, realtype cj)
- void fJvB (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector vB, N_Vector JvB, realtype cj)
- virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root) override
- void froot (realtype t, N_Vector x, N_Vector dx, realtype *root)
- virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot) override
- void fxdot (realtype t, N_Vector x, N_Vector dx, N_Vector xdot)
- void fxBdot (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector xBdot)
- void fqBdot (realtype t, N_Vector x, N_Vector dx, N_Vector xB, N_Vector dxB, N_Vector qBdot)
- void fdxdotdp (const realtype t, const N_Vector x, const N_Vector dx)
- virtual void fdxdotdp (realtype t, AmiVector *x, AmiVector *dx) override
- void fsxdot (realtype t, N_Vector x, N_Vector dx, int ip, N_Vector sx, N_Vector sdx, N_Vector sxdot)
- void fM (realtype t, const N_Vector x)

  Mass matrix for DAE systems.
- virtual std::unique_ptr< Solver > getSolver () override

Protected Member Functions

- virtual void fJ (realtype *J, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)=0
- virtual void fJB (realtype *JB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *w, const realtype *dwdx)
- virtual void fJSparse (SlsMat JSparse, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)=0
- virtual void fJSparseB (SlsMat JSparseB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *w, const realtype *dwdx)
- virtual void fJDiag (realtype *JDiag, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *w, const realtype *dwdx)
- virtual void fJv (realtype *Jv, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *dx, const realtype *v, const realtype *w, const realtype *dwdx)
- virtual void fJvB (realtype *JvB, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype cj, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *vB, const realtype *w, const realtype *dwdx)
- virtual void froot (realtype *root, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype *dx)
- virtual void fxdot (realtype *xdot, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype *dx, const realtype *w)=0
- virtual void fxBdot (realtype *xBdot, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *w, const realtype *dwdx)
- virtual void fqBdot (realtype *qBdot, const int ip, const realtype t, const realtype *x, const double *p, const double *k, const realtype *h, const realtype *xB, const realtype *dx, const realtype *dxB, const realtype *w, const realtype *dwdp)
- virtual void fdxdotdp (realtype *dxdotdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *dx, const realtype *w, const realtype *dwdp)
- virtual void fsxdot (realtype *sxdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *dx, const realtype *sx, const realtype *sdx, const realtype *w, const realtype *dwdx, const realtype *M, const realtype *J, const realtype *dxdotdp)
- virtual void fM (realtype *M, const realtype t, const realtype *x, const realtype *p, const realtype *k)

Additional Inherited Members

10.11.1 Detailed Description

Definition at line 24 of file model_dae.h.

10.11.2 Constructor & Destructor Documentation

10.11.2.1 Model_DAE() [1/2]

```
Model_DAE ( )
```

default constructor

Definition at line 27 of file model_dae.h.

10.11.2.2 Model_DAE() [2/2]

```
Model_DAE (
            const int nx,
            const int nxtrue,
            const int ny,
            const int nytrue,
            const int nz,
            const int nztrue,
            const int ne,
            const int nJ,
            const int nw,
            const int ndwdx,
            const int ndwdp,
            const int nnz,
            const int ubw,
            const int lbw,
            const AMICI_o2mode o2mode,
            const std::vector< realtype > p,
            const std::vector< realtype > k,
            const std::vector< int > plist,
            const std::vector< realtype > idlist,
            const std::vector< int > z2event )
```

constructor with model dimensions

Parameters

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

Definition at line 53 of file model_dae.h.

10.11.3 Member Function Documentation

10.11.3.1 fJ() [1/3]

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

Dense Jacobian function

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xdot	values of residual function (unused)
J	dense matrix to which values of the jacobian will be written

Implements Model.

Definition at line 6 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.2 fJ() [2/3]

```
void fJ (
            realtype t,
            realtype cj,
            N_Vector x,
            N_Vector dx,
            N_Vector xdot,
            DlsMat J )
```

Jacobian of xdot with respect to states x

Parameters

t	timepoint
cj	scaling factor, inverse of the step size
x	Vector with the states
dx	Vector with the derivative states
xdot	Vector with the right hand side
J	Matrix to which the Jacobian will be written

Definition at line 20 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.3 fJB() [1/2]

```
void fJB (
            realtype t,
            realtype cj,
            N_Vector x,
            N_Vector dx,
            N_Vector xB,
            N_Vector dxB,
            DlsMat JB )
```

Jacobian of xBdot with respect to adjoint state xB

Parameters

t	timepoint
cj	scaling factor, inverse of the step size
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
JB	Matrix to which the Jacobian will be written

Definition at line 158 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.4 fJSparse() [1/3]

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

Sparse Jacobian function

Parameters

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

Implements Model.

Definition at line 28 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.5 fJSparse() [2/3]

```
void fJSparse (
            realtype t,
            realtype cj,
            N_Vector x,
            N_Vector dx,
            SlsMat J )
```

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

Parameters

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

Definition at line 40 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.6 fJSparseB() [1/2]

```
void fJSparseB (
            realtype t,
            realtype cj,
            N_Vector x,
            N_Vector dx,
            N_Vector xB,
            N_Vector dxB,
            SlsMat JB )
```

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

Parameters

t	timepoint
cj	scalar in Jacobian
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
JB	Matrix to which the Jacobian will be written

Definition at line 175 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.7 fJDiag() [1/2]

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

diagonalized Jacobian (for preconditioning)

Parameters

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

Returns

status flag indicating successful execution

Implements Model.

Definition at line 111 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.8 fJv() [1/3]

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

Jacobian multiply function

Parameters

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

Implements Model.

Definition at line 47 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.9 fJv() [2/3]

```
void fJv (
            realtype t,
            N_Vector x,
            N_Vector dx,
            N_Vector v,
            N_Vector Jv,
            realtype cj )
```

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

Parameters

t	timepoint **Type**: realtype
cj	scaling factor, inverse of the step size
x	Vector with the states
dx	Vector with the derivative states
v	Vector with which the Jacobian is multiplied
Jv	Vector to which the Jacobian vector product will be written

Definition at line 62 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.10 fJvB() [1/2]

```
void fJvB (
            realtype t,
            N_Vector x,
            N_Vector dx,
            N_Vector xB,
            N_Vector dxB,
            N_Vector vB,
            N_Vector JvB,
            realtype cj )
```

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

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
vB	Vector with which the Jacobian is multiplied
JvB	Vector to which the Jacobian vector product will be written
cj	scalar in Jacobian (inverse stepsize)

Definition at line 194 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.11 froot() [1/3]

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

Root function

Parameters

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

Implements Model.

Definition at line 70 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.12 froot() [2/3]

```
void froot (
            realtype t,
            N_Vector x,
            N_Vector dx,
            realtype * root )
```

Event trigger function for events

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
root	array with root function values

Definition at line 80 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.13 fxdot() [1/3]

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

Residual function

Parameters

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

Implements Model.

Definition at line 86 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.14 fxdot() [2/3]

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

residual function of the DAE

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
xdot	Vector with the right hand side

Definition at line 96 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.15 fxBdot() [1/2]

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

Right hand side of differential equation for adjoint state xB

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
xBdot	Vector with the adjoint right hand side

Definition at line 211 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.16 fqBdot() [1/2]

```
void fqBdot (
            realtype t,
            N_Vector x,
            N_Vector dx,
            N_Vector xB,
            N_Vector dxB,
            N_Vector qBdot )
```

Right hand side of integral equation for quadrature states qB

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
xB	Vector with the adjoint states
dxB	Vector with the adjoint derivative states
qBdot	Vector with the adjoint quadrature right hand side

Definition at line 228 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.17 fdxdotdp() [1/3]

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

Sensitivity of dx/dt wrt model parameters p

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states

Returns

status flag indicating successful execution

Definition at line 127 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.18 fdxdotdp() [2/3]

```
virtual void fdxdotdp (
            realtype t,
            AmiVector * x,
            AmiVector * dx )  [override], [virtual]
```

parameter derivative of residual function

Parameters

t	time
x	state
dx	time derivative of state (DAE only)

Returns

flag indicating successful evaluation

Implements Model.

Definition at line 95 of file model_dae.h.

Here is the call graph for this function:

10.11.3.19 fsxdot() [1/2]

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

Right hand side of differential equation for state sensitivities sx

Parameters

t	timepoint
x	Vector with the states
dx	Vector with the derivative states
ip	parameter index
sx	Vector with the state sensitivities
sdx	Vector with the derivative state sensitivities
sxdot	Vector with the sensitivity right hand side

Definition at line 247 of file model_dae.cpp.

Here is the call graph for this function:

10.11.3.20 fM() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 140 of file model_dae.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.11.3.21 getSolver()

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

Retrieves the solver object

Returns

The Solver instance

Implements Model.

Definition at line 145 of file model_dae.cpp.

10.11.3.22 fJ() [3/3]

```
virtual void fJ (
            realtype * J,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * dx,
            const realtype * w,
            const realtype * dwdx )  [protected], [pure virtual]
```

model specific implementation for fJ

Parameters

J	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
dx	Vector with the derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

10.11.3.23 fJB() [2/2]

```
virtual void fJB (
            realtype * JB,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * xB,
            const realtype * dx,
            const realtype * dxB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJB

Parameters

JB	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states

Parameters

p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
xB	Vector with the adjoint states
dx	Vector with the derivative states
dxB	Vector with the adjoint derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 137 of file model_dae.h.

10.11.3.24 fJSparse() [3/3]

```
virtual void fJSparse (
            SlsMat JSparse,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * dx,
            const realtype * w,
            const realtype * dwdx )  [protected], [pure virtual]
```

model specific implementation for fJSparse

Parameters

JSparse	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
dx	Vector with the derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

10.11.3.25 fJSparseB() [2/2]

```
virtual void fJSparseB (
            SlsMat JSparseB,
```

```
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * xB,
            const realtype * dx,
            const realtype * dxB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJSparseB

Parameters

JSparseB	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
xB	Vector with the adjoint states
dx	Vector with the derivative states
dxB	Vector with the adjoint derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 172 of file model_dae.h.

10.11.3.26 fJDiag() [2/2]

```
virtual void fJDiag (
            realtype * JDiag,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype cj,
            const realtype * dx,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJDiag

Parameters

JDiag	array to which the Jacobian diagonal will be written
t	timepoint
x	Vector with the states

Parameters

p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
dx	Vector with the derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 190 of file model_dae.h.

10.11.3.27 fJv() [3/3]

```
virtual void fJv (
            realtype * Jv,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * dx,
            const realtype * v,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJv

Parameters

Jv	Matrix vector product of J with a vector v
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
dx	Vector with the derivative states
v	Vector with which the Jacobian is multiplied
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 208 of file model_dae.h.

10.11.3.28 fJvB() [2/2]

```
virtual void fJvB (
            realtype * JvB,
```

```
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype cj,
            const realtype * xB,
            const realtype * dx,
            const realtype * dxB,
            const realtype * vB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJvB

Parameters

JvB	Matrix vector product of JB with a vector v
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
cj	scaling factor, inverse of the step size
xB	Vector with the adjoint states
dx	Vector with the derivative states
dxB	Vector with the adjoint derivative states
vB	Vector with which the Jacobian is multiplied
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 229 of file model_dae.h.

10.11.3.29 froot() [3/3]

```
virtual void froot (
            realtype * root,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype * dx )  [protected], [virtual]
```

model specific implementation for froot

Parameters

root	values of the trigger function
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
dx	Vector with the derivative states

Definition at line 244 of file model_dae.h.

10.11.3.30 fxdot() [3/3]

```
virtual void fxdot (
            realtype * xdot,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype * dx,
            const realtype * w )  [protected], [pure virtual]
```

model specific implementation for fxdot

Parameters

xdot	residual function
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables
dx	Vector with the derivative states

10.11.3.31 fxBdot() [2/2]

```
virtual void fxBdot (
            realtype * xBdot,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype * xB,
            const realtype * dx,
            const realtype * dxB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fxBdot

Parameters

xBdot	adjoint residual function
t	timepoint
x	Vector with the states
p	parameter vector

Parameters

k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
dx	Vector with the derivative states
dxB	Vector with the adjoint derivative states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 275 of file model_dae.h.

10.11.3.32 fqBdot() [2/2]

```
virtual void fqBdot (
            realtype * qBdot,
            const int ip,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype * xB,
            const realtype * dx,
            const realtype * dxB,
            const realtype * w,
            const realtype * dwdp )  [protected], [virtual]
```

model specific implementation for fqBdot

Parameters

qBdot	adjoint quadrature equation
ip	sensitivity index
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
dx	Vector with the derivative states
dxB	Vector with the adjoint derivative states
w	vector with helper variables
dwdp	derivative of w wrt p

Definition at line 295 of file model_dae.h.

10.11.3.33 fdxdotdp() [3/3]

```
virtual void fdxdotdp (
            realtype * dxdotdp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip,
            const realtype * dx,
            const realtype * w,
            const realtype * dwdp )  [protected], [virtual]
```

model specific implementation of fdxdotdp

Parameters

dxdotdp	partial derivative xdot wrt p
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
ip	parameter index
dx	Vector with the derivative states
w	vector with helper variables
dwdp	derivative of w wrt p

Definition at line 313 of file model_dae.h.

10.11.3.34 fsxdot() [2/2]

```
virtual void fsxdot (
            realtype * sxdot,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip,
            const realtype * dx,
            const realtype * sx,
            const realtype * sdx,
            const realtype * w,
            const realtype * dwdx,
            const realtype * M,
            const realtype * J,
            const realtype * dxdotdp )  [protected], [virtual]
```

model specific implementation of fsxdot

Parameters

sxdot	sensitivity rhs
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
ip	parameter index
dx	Vector with the derivative states
sx	Vector with the state sensitivities
sdx	Vector with the derivative state sensitivities
w	vector with helper variables
dwdx	derivative of w wrt x
M	mass matrix
J	jacobian
dxdotdp	parameter derivative of residual function

Definition at line 335 of file model_dae.h.

10.11.3.35 fM() [2/2]

```
virtual void fM (
            realtype * M,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k )  [protected], [virtual]
```

model specific implementation of fM

Parameters

M	mass matrix
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector

Definition at line 349 of file model_dae.h.

10.12 Model_ODE Class Reference

The Model class represents an AMICI ODE model. The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchroneously.

```
#include <model_ode.h>
```

Inheritance diagram for Model_ODE:

Public Member Functions

- Model_ODE ()
- Model_ODE (const int nx, const int nxtrue, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int nnz, const int ubw, const int lbw, const AMICI_o2mode o2mode, const std::vector< realtype > p, const std::vector< realtype > k, const std::vector< int > plist, const std::vector< realtype > idlist, const std::vector< int > z2event)
- Model_ODE (Model_ODE const &other)

  Copy constructor.
- virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J) override
- void fJ (realtype t, N_Vector x, N_Vector xdot, DlsMat J)
- void fJB (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, DlsMat JB)
- virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J) override
- void fJSparse (realtype t, N_Vector x, SlsMat J)
- void fJSparseB (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot, SlsMat JB)
- void fJDiag (realtype t, N_Vector JDiag, N_Vector x)
- virtual void fJDiag (realtype t, AmiVector *Jdiag, realtype cj, AmiVector *x, AmiVector *dx) override
- virtual void fJv (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot, AmiVector *v, AmiVector *nJv, realtype cj) override
- void fJv (N_Vector v, N_Vector Jv, realtype t, N_Vector x)
- void fJvB (N_Vector vB, N_Vector JvB, realtype t, N_Vector x, N_Vector xB)
- virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root) override
- void froot (realtype t, N_Vector x, realtype *root)
- virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot) override
- void fxdot (realtype t, N_Vector x, N_Vector xdot)
- void fxBdot (realtype t, N_Vector x, N_Vector xB, N_Vector xBdot)
- void fqBdot (realtype t, N_Vector x, N_Vector xB, N_Vector qBdot)
- void fdxdotdp (const realtype t, const N_Vector x)
- virtual void fdxdotdp (realtype t, AmiVector *x, AmiVector *dx) override
- void fsxdot (realtype t, N_Vector x, int ip, N_Vector sx, N_Vector sxdot)
- virtual std::unique_ptr< Solver > getSolver () override

Protected Member Functions

- virtual void fJ (realtype *J, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)=0
- virtual void fJB (realtype *JB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void fJSparse (SlsMat JSparse, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)=0
- virtual void fJSparseB (SlsMat JSparseB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void fJDiag (realtype *JDiag, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const realtype *dwdx)
- virtual void fJv (realtype *Jv, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *v, const realtype *w, const realtype *dwdx)
- virtual void fJvB (realtype *JvB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *vB, const realtype *w, const realtype *dwdx)
- virtual void froot (realtype *root, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fxdot (realtype *xdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)=0
- virtual void fxBdot (realtype *xBdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdx)
- virtual void fqBdot (realtype *qBdot, const int ip, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *xB, const realtype *w, const realtype *dwdp)
- virtual void fdxdotdp (realtype *dxdotdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *w, const realtype *dwdp)
- virtual void fsxdot (realtype *sxdot, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const realtype *sx, const realtype *w, const realtype *dwdx, const realtype *J, const realtype *dxdotdp)

Additional Inherited Members

10.12.1 Detailed Description

Definition at line 23 of file model_ode.h.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 Model_ODE() [1/3]

`Model_ODE ( )`

default constructor

Definition at line 26 of file model_ode.h.

10.12.2.2 Model_ODE() [2/3]

```
Model_ODE (
            const int nx,
            const int nxtrue,
            const int ny,
            const int nytrue,
            const int nz,
            const int nztrue,
            const int ne,
            const int nJ,
            const int nw,
            const int ndwdx,
            const int ndwdp,
            const int nnz,
            const int ubw,
            const int lbw,
            const AMICI_o2mode o2mode,
            const std::vector< realtype > p,
            const std::vector< realtype > k,
            const std::vector< int > plist,
            const std::vector< realtype > idlist,
            const std::vector< int > z2event )
```

constructor with model dimensions

Parameters

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

Definition at line 52 of file model_ode.h.

10.12.2.3 Model_ODE() [3/3]

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

Parameters

other	

Definition at line 66 of file model_ode.h.

10.12.3 Member Function Documentation

10.12.3.1 fJ() [1/3]

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

Dense Jacobian function

Parameters

t	time
cj	scaling factor (inverse of timestep, DAE only)
x	state
dx	time derivative of state (DAE only)
xdot	values of residual function (unused)
J	dense matrix to which values of the jacobian will be written

Implements Model.

Definition at line 6 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.2 fJ() [2/3]

```
void fJ (
            realtype t,
            N_Vector x,
            N_Vector xdot,
            DlsMat J )
```

implementation of fJ at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the AmiVector level implementation

Parameters

t	timepoint
x	Vector with the states
xdot	Vector with the right hand side
J	Matrix to which the Jacobian will be written

Definition at line 20 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.3 fJB() [1/2]

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

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

Parameters

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

Definition at line 141 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.4 fJSparse() [1/3]

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

Sparse Jacobian function

Parameters

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

Implements Model.

Definition at line 27 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.5 fJSparse() [2/3]

```
void fJSparse (
            realtype t,
            N_Vector x,
            SlsMat J )
```

implementation of fJSparse at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the AmiVector level implementation

Parameters

t	timepoint
x	Vector with the states
J	Matrix to which the Jacobian will be written

Definition at line 39 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.6 fJSparseB() [1/2]

```
void fJSparseB (
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector xBdot,
            SlsMat JB )
```

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

Parameters

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

Definition at line 157 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.7 fJDiag() [1/3]

```
void fJDiag (
            realtype t,
            N_Vector JDiag,
            N_Vector x )
```

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

Parameters

t	timepoint
JDiag	Vector to which the Jacobian diagonal will be written
x	Vector with the states

Definition at line 170 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.8 fJDiag() [2/3]

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

diagonalized Jacobian (for preconditioning)

Parameters

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

Returns

status flag indicating successful execution

Implements Model.

Definition at line 109 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.9 fJv() [1/3]

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

Jacobian multiply function

Parameters

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

Implements Model.

Definition at line 46 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.10 fJv() [2/3]

```
void fJv (
            N_Vector v,
            N_Vector Jv,
            realtype t,
            N_Vector x )
```

implementation of fJv at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the AmiVector level implementation

Parameters

t	timepoint
x	Vector with the states
v	Vector with which the Jacobian is multiplied
Jv	Vector to which the Jacobian vector product will be written

Definition at line 60 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.11 fJvB() [1/2]

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

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

Parameters

t	timepoint
x	Vector with the states
xB	Vector with the adjoint states
vB	Vector with which the Jacobian is multiplied
JvB	Vector to which the Jacobian vector product will be written

Definition at line 186 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.12 froot() [1/3]

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

Root function

Parameters

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

Implements Model.

Definition at line 67 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.13 froot() [2/3]

```
void froot (
            realtype t,
            N_Vector x,
            realtype * root )
```

implementation of froot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the AmiVector level implementation

Parameters

t	timepoint
x	Vector with the states
root	array with root function values

Definition at line 78 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.14 fxdot() [1/3]

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

Residual function

Parameters

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

Implements Model.

Definition at line 83 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.15 fxdot() [2/3]

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

implementation of fxdot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation aswell as the AmiVector level implementation

Parameters

t	timepoint
x	Vector with the states
xdot	Vector with the right hand side

Definition at line 94 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.16 fxBdot() [1/2]

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

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

Parameters

t	timepoint
x	Vector with the states
xB	Vector with the adjoint states
xBdot	Vector with the adjoint right hand side

Definition at line 200 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.17 fqBdot() [1/2]

```
void fqBdot (
            realtype t,
            N_Vector x,
            N_Vector xB,
            N_Vector qBdot )
```

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

Parameters

t	timepoint
x	Vector with the states
xB	Vector with the adjoint states
qBdot	Vector with the adjoint quadrature right hand side

Definition at line 214 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.18 fdxdotdp() [1/3]

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

Sensitivity of dx/dt wrt model parameters p

Parameters

t	timepoint
x	Vector with the states

Returns

status flag indicating successful execution

Definition at line 121 of file model_ode.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.12.3.19 fdxdotdp() [2/3]

```
virtual void fdxdotdp (
            realtype t,
            AmiVector * x,
            AmiVector * dx )  [override], [virtual]
```

parameter derivative of residual function

Parameters

t	time
x	state
dx	time derivative of state (DAE only)

Returns

flag indicating successful evaluation

Implements Model.

Definition at line 104 of file model_ode.h.

Here is the call graph for this function:

10.12.3.20 fsxdot() [1/2]

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

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

Parameters

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

Definition at line 231 of file model_ode.cpp.

Here is the call graph for this function:

10.12.3.21 getSolver()

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

Retrieves the solver object

Returns

The Solver instance

Implements Model.

Definition at line 129 of file model_ode.cpp.

10.12.3.22 fJ() [3/3]

```
virtual void fJ (
            realtype * J,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w,
            const realtype * dwdx ) [protected], [pure virtual]
```

model specific implementation for fJ

Parameters

J	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

10.12.3.23 fJB() [2/2]

```
virtual void fJB (
            realtype * JB,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * xB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJB

Parameters

JB	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 137 of file model_ode.h.

10.12.3.24 fJSparse() [3/3]

```
virtual void fJSparse (
            SlsMat JSparse,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w,
            const realtype * dwdx )  [protected], [pure virtual]
```

model specific implementation for fJSparse

Parameters

JSparse	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables
dwdx	derivative of w wrt x

10.12.3.25 fJSparseB() [2/2]

```
virtual void fJSparseB (
            SlsMat JSparseB,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * xB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJSparseB

Parameters

JSparseB	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 165 of file model_ode.h.

10.12.3.26 fJDiag() [3/3]

```
virtual void fJDiag (
            realtype * JDiag,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJDiag

Parameters

JDiag	Matrix to which the Jacobian will be written
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 180 of file model_ode.h.

10.12.3.27 fJv() [3/3]

```
virtual void fJv (
            realtype * Jv,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * v,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJv

Parameters

Jv	Matrix vector product of J with a vector v
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
v	Vector with which the Jacobian is multiplied
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 196 of file model_ode.h.

10.12.3.28 fJvB() [2/2]

```
virtual void fJvB (
            realtype * JvB,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * xB,
            const realtype * vB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fJvB

Parameters

JvB	Matrix vector product of JB with a vector v

Parameters

t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
vB	Vector with which the Jacobian is multiplied
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 213 of file model_ode.h.

10.12.3.29 froot() [3/3]

```
virtual void froot (
            realtype * root,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [protected], [virtual]
```

model specific implementation for froot

Parameters

root	values of the trigger function
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector

Definition at line 226 of file model_ode.h.

10.12.3.30 fxdot() [3/3]

```
virtual void fxdot (
            realtype * xdot,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w )  [protected], [pure virtual]
```

model specific implementation for fxdot

Parameters

xdot	residual function
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
w	vector with helper variables

10.12.3.31 fxBdot() [2/2]

```
virtual void fxBdot (
            realtype * xBdot,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * xB,
            const realtype * w,
            const realtype * dwdx )  [protected], [virtual]
```

model specific implementation for fxBdot

Parameters

xBdot	adjoint residual function
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
w	vector with helper variables
dwdx	derivative of w wrt x

Definition at line 252 of file model_ode.h.

10.12.3.32 fqBdot() [2/2]

```
virtual void fqBdot (
            realtype * qBdot,
            const int ip,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
```

```
            const realtype * h,
            const realtype * xB,
            const realtype * w,
            const realtype * dwdp )  [protected], [virtual]
```

model specific implementation for fqBdot

Parameters

qBdot	adjoint quadrature equation
ip	sensitivity index
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
xB	Vector with the adjoint states
w	vector with helper variables
dwdp	derivative of w wrt p

Definition at line 269 of file model_ode.h.

10.12.3.33 fdxdotdp() [3/3]

```
virtual void fdxdotdp (
            realtype * dxdotdp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip,
            const realtype * w,
            const realtype * dwdp )  [protected], [virtual]
```

model specific implementation of fdxdotdp

Parameters

dxdotdp	partial derivative xdot wrt p
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
ip	parameter index
w	vector with helper variables
dwdp	derivative of w wrt p

Definition at line 285 of file model_ode.h.

10.12.3.34 fsxdot() [2/2]

```
virtual void fsxdot (
            realtype * sxdot,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip,
            const realtype * sx,
            const realtype * w,
            const realtype * dwdx,
            const realtype * J,
            const realtype * dxdotdp )  [protected], [virtual]
```

model specific implementation of fsxdot

Parameters

sxdot	sensitivity rhs
t	timepoint
x	Vector with the states
p	parameter vector
k	constants vector
h	heavyside vector
ip	parameter index
sx	Vector with the state sensitivities
w	vector with helper variables
dwdx	derivative of w wrt x
J	jacobian
dxdotdp	parameter derivative of residual function

Definition at line 304 of file model_ode.h.

10.13 NewtonFailure Class Reference

newton failure exception this exception should be thrown when the steady state computation failed to converge for this exception we can assume that we can recover from the exception and return a solution struct to the user

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

Inheritance diagram for NewtonFailure:

Public Member Functions

- NewtonFailure (const char *msg)

10.13.1 Detailed Description

Definition at line 179 of file exception.h.

10.13.2 Constructor & Destructor Documentation

10.13.2.1 NewtonFailure()

```
NewtonFailure (
            const char * msg )
```

constructor, simply calls AmiException constructor

Parameters

in	*msg*	

Definition at line 184 of file exception.h.

10.14 NewtonSolver Class Reference

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

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

Inheritance diagram for NewtonSolver:

Public Member Functions

- NewtonSolver (realtype ∗t, AmiVector ∗x, Model ∗model, ReturnData ∗rdata)
- void getStep (int ntry, int nnewt, AmiVector ∗delta)
- void getSensis (const int it, AmiVectorArray ∗sx)
- virtual void prepareLinearSystem (int ntry, int nnewt)=0
- virtual void solveLinearSystem (AmiVector ∗rhs)=0

Static Public Member Functions

- static NewtonSolver ∗ getSolver (realtype ∗t, AmiVector ∗x, int linsolType, Model ∗model, ReturnData ∗rdata, int maxlinsteps, int maxsteps, double atol, double rtol)

Public Attributes

- int maxlinsteps = 0
- int maxsteps = 0
- double atol = 1e-16
- double rtol = 1e-8

Protected Attributes

- realtype ∗ t
- Model ∗ model
- ReturnData ∗ rdata
- AmiVector xdot
- AmiVector ∗ x
- AmiVector dx

10.14.1 Detailed Description

Definition at line 22 of file newton_solver.h.

10.14.2 Constructor & Destructor Documentation

10.14.2.1 NewtonSolver()

```
NewtonSolver (
            realtype * t,
            AmiVector * x,
            Model * model,
            ReturnData * rdata )
```

default constructor, initializes all members with the provided objects

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the AMICI model object
rdata	pointer to the return data object

Definition at line 18 of file newton_solver.cpp.

10.14.3 Member Function Documentation

10.14.3.1 getSolver()

```
NewtonSolver * getSolver (
            realtype * t,
            AmiVector * x,
            int linsolType,
            Model * model,
            ReturnData * rdata,
            int maxlinsteps,
            int maxsteps,
            double atol,
            double rtol )  [static]
```

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

Parameters

t	pointer to time variable
x	pointer to state variables
linsolType	integer indicating which linear solver to use
model	pointer to the AMICI model object
rdata	pointer to the return data object
maxlinsteps	maximum number of allowed linear steps per Newton step for steady state computation
maxsteps	maximum number of allowed Newton steps for steady state computation
atol	absolute tolerance
rtol	relative tolerance

Returns

solver NewtonSolver according to the specified linsolType

Definition at line 36 of file newton_solver.cpp.

Here is the caller graph for this function:

10.14.3.2 getStep()

```
void getStep (
            int ntry,
            int nnewt,
            AmiVector * delta )
```

Computes the solution of one Newton iteration

Parameters

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

Definition at line 107 of file newton_solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.14.3.3 getSensis()

```
void getSensis (
            const int it,
            AmiVectorArray * sx )
```

Computes steady state sensitivities

Parameters

it	integer index of current time step
sx	pointer to state variable sensitivities

Definition at line 127 of file newton_solver.cpp.

Here is the call graph for this function:

10.14.3.4 prepareLinearSystem()

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

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

Parameters

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

Implemented in NewtonSolverIterative, NewtonSolverSparse, and NewtonSolverDense.

Here is the caller graph for this function:

10.14.3.5 solveLinearSystem()

```
virtual void solveLinearSystem (
            AmiVector * rhs )  [pure virtual]
```

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implemented in NewtonSolverIterative, NewtonSolverSparse, and NewtonSolverDense.

Here is the caller graph for this function:

10.14.4 Member Data Documentation

10.14.4.1 maxlinsteps

```
int maxlinsteps = 0
```

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

Definition at line 56 of file newton_solver.h.

10.14.4.2 maxsteps

```
int maxsteps = 0
```

maximum number of allowed Newton steps for steady state computation

Definition at line 58 of file newton_solver.h.

10.14.4.3 atol

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

absolute tolerance

Definition at line 60 of file newton_solver.h.

10.14.4.4 rtol

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

relative tolerance

Definition at line 62 of file newton_solver.h.

10.14.4.5 t

```
realtype* t  [protected]
```

time variable

Definition at line 66 of file newton_solver.h.

10.14.4.6 model

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

pointer to the AMICI model object

Definition at line 68 of file newton_solver.h.

10.14.4.7 rdata

```
ReturnData* rdata [protected]
```

pointer to the return data object

Definition at line 70 of file newton_solver.h.

10.14.4.8 xdot

```
AmiVector xdot [protected]
```

right hand side AmiVector

Definition at line 72 of file newton_solver.h.

10.14.4.9 x

```
AmiVector* x [protected]
```

current state

Definition at line 74 of file newton_solver.h.

10.14.4.10 dx

```
AmiVector dx [protected]
```

current state time derivative (DAE)

Definition at line 76 of file newton_solver.h.

10.15 NewtonSolverDense Class Reference

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

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

Inheritance diagram for NewtonSolverDense:

Public Member Functions

- NewtonSolverDense (realtype ∗t, AmiVector ∗x, Model ∗model, ReturnData ∗rdata)
- void solveLinearSystem (AmiVector ∗rhs) override
- void prepareLinearSystem (int ntry, int nnewt) override

Additional Inherited Members

10.15.1 Detailed Description

Definition at line 85 of file newton_solver.h.

10.15.2 Constructor & Destructor Documentation

10.15.2.1 NewtonSolverDense()

```
NewtonSolverDense (
            realtype * t,
            AmiVector * x,
            Model * model,
            ReturnData * rdata )
```

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

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the AMICI model object
rdata	pointer to the return data object

Definition at line 161 of file newton_solver.cpp.

10.15.3 Member Function Documentation

10.15.3.1 solveLinearSystem()

```
void solveLinearSystem (
            AmiVector * rhs )  [override], [virtual]
```

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implements NewtonSolver.

Definition at line 200 of file newton_solver.cpp.

Here is the call graph for this function:

10.15.3.2 prepareLinearSystem()

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

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

Parameters

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

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

Parameters

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

Implements NewtonSolver.

Definition at line 180 of file newton_solver.cpp.

Here is the call graph for this function:

10.16 NewtonSolverIterative Class Reference

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

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

Inheritance diagram for NewtonSolverIterative:

Public Member Functions

- NewtonSolverIterative (realtype ∗t, AmiVector ∗x, Model ∗model, ReturnData ∗rdata)
- void solveLinearSystem (AmiVector ∗rhs)
- void prepareLinearSystem (int ntry, int nnewt)
- void linsolveSPBCG (int ntry, int nnewt, AmiVector ∗ns_delta)

Additional Inherited Members

10.16.1 Detailed Description

Definition at line 131 of file newton_solver.h.

10.16.2 Constructor & Destructor Documentation

10.16.2.1 NewtonSolverIterative()

```
NewtonSolverIterative (
            realtype * t,
            AmiVector * x,
            Model * model,
            ReturnData * rdata )
```

default constructor, initializes all members with the provided objects

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the AMICI model object
rdata	pointer to the return data object

Definition at line 324 of file newton_solver.cpp.

10.16.3 Member Function Documentation

10.16.3.1 solveLinearSystem()

```
void solveLinearSystem (
            AmiVector * rhs )  [virtual]
```

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

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

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Implements NewtonSolver.

Definition at line 362 of file newton_solver.cpp.

Here is the call graph for this function:

10.16.3.2 prepareLinearSystem()

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

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

Parameters

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

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

Parameters

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

Implements NewtonSolver.

Definition at line 341 of file newton_solver.cpp.

10.16.3.3 linsolveSPBCG()

```
void linsolveSPBCG (
            int ntry,
            int nnewt,
            AmiVector * ns_delta )
```

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

Parameters

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

Definition at line 375 of file newton_solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.17 NewtonSolverSparse Class Reference

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

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

Inheritance diagram for NewtonSolverSparse:

Public Member Functions

- NewtonSolverSparse (realtype ∗t, AmiVector ∗x, Model ∗model, ReturnData ∗rdata)
- void solveLinearSystem (AmiVector ∗rhs) override
- void prepareLinearSystem (int ntry, int nnewt) override

Additional Inherited Members

10.17.1 Detailed Description

Definition at line 105 of file newton_solver.h.

10.17.2 Constructor & Destructor Documentation

10.17.2.1 NewtonSolverSparse()

```
NewtonSolverSparse (
            realtype * t,
            AmiVector * x,
            Model * model,
            ReturnData * rdata )
```

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

Parameters

t	pointer to time variable
x	pointer to state variables
model	pointer to the AMICI model object
rdata	pointer to the return data object

Definition at line 230 of file newton_solver.cpp.

10.17.3 Member Function Documentation

10.17.3.1 solveLinearSystem()

```
void solveLinearSystem (
            AmiVector * rhs )  [override], [virtual]
```

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system, will be overwritten by solution to the linear system

Solves the linear system for the Newton step

Parameters

rhs	containing the RHS of the linear system,will be overwritten by solution to the linear system

Implements NewtonSolver.

Definition at line 290 of file newton_solver.cpp.

Here is the call graph for this function:

10.17.3.2 prepareLinearSystem()

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

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

Parameters

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

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

Parameters

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

Implements NewtonSolver.

Definition at line 250 of file newton_solver.cpp.

Here is the call graph for this function:

10.18 ReturnData Class Reference

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

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

Public Member Functions

- ReturnData ()

  default constructor
- ReturnData (Solver const &solver, const Model ∗model)
- void initializeObjectiveFunction ()

  initializeObjectiveFunction
- void invalidate (const realtype t)
- void invalidateLLH ()
- void applyChainRuleFactorToSimulationResults (const Model ∗model)

Public Attributes

- std::vector< realtype > ts
- std::vector< realtype > xdot
- std::vector< realtype > J
- std::vector< realtype > z
- std::vector< realtype > sigmaz
- std::vector< realtype > sz
- std::vector< realtype > ssigmaz
- std::vector< realtype > rz
- std::vector< realtype > srz
- std::vector< realtype > s2rz
- std::vector< realtype > x
- std::vector< realtype > sx
- std::vector< realtype > y
- std::vector< realtype > sigmay
- std::vector< realtype > sy
- std::vector< realtype > ssigmay
- std::vector< realtype > res
- std::vector< realtype > sres
- std::vector< int > numsteps
- std::vector< int > numstepsB
- std::vector< int > numrhsevals
- std::vector< int > numrhsevalsB
- std::vector< int > numerrtestfails
- std::vector< int > numerrtestfailsB
- std::vector< int > numnonlinsolvconvfails
- std::vector< int > numnonlinsolvconvfailsB
- std::vector< int > order
- int newton_status = 0
- double newton_time = 0.0
- std::vector< int > newton_numsteps
- std::vector< int > newton_numlinsteps
- std::vector< realtype > x0
- std::vector< realtype > sx0
- realtype llh = 0.0
- realtype chi2 = 0.0
- std::vector< realtype > sllh
- std::vector< realtype > s2llh
- int status = 0
- const int np
- const int nk
- const int nx
- const int nxtrue
- const int ny
- const int nytrue
- const int nz
- const int nztrue
- const int ne
- const int nJ
- const int nplist
- const int nmaxevent
- const int nt
- const int newton_maxsteps
- std::vector< AMICI_parameter_scaling > pscale
- const AMICI_o2mode o2mode
- const AMICI_sensi_order sensi
- const AMICI_sensi_meth sensi_meth

Friends

- template<class Archive >
  void boost::serialization::serialize (Archive &ar, ReturnData &r, const unsigned int version)

  Serialize ReturnData (see boost::serialization::serialize)

10.18.1 Detailed Description

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

Definition at line 28 of file rdata.h.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 ReturnData()

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

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

Parameters

solver	solver
model	pointer to model specification object bool

Definition at line 22 of file rdata.cpp.

Here is the call graph for this function:

10.18.3 Member Function Documentation

10.18.3.1 invalidate()

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

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

Parameters

t	time of integration failure

Definition at line 100 of file rdata.cpp.

Here is the call graph for this function:

10.18.3.2 invalidateLLH()

```
void invalidateLLH ( )
```

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

Definition at line 139 of file rdata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.18.3.3 applyChainRuleFactorToSimulationResults()

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

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

Parameters

model	Model from which the ReturnData was obtained

Definition at line 151 of file rdata.cpp.

Here is the call graph for this function:

10.18.4 Friends And Related Function Documentation

10.18.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
            Archive & ar,
            ReturnData & r,
            const unsigned int version )  [friend]
```

Parameters

ar	Archive to serialize to
r	Data to serialize
version	Version number

10.18.5 Member Data Documentation

10.18.5.1 ts

```
std::vector<realtype> ts
```

timepoints (dimension: nt)

Definition at line 48 of file rdata.h.

10.18.5.2 xdot

```
std::vector<realtype> xdot
```

time derivative (dimension: nx)

Definition at line 51 of file rdata.h.

10.18.5.3 J

```
std::vector<realtype> J
```

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

Definition at line 55 of file rdata.h.

10.18.5.4 z

```
std::vector<realtype> z
```

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

Definition at line 58 of file rdata.h.

10.18.5.5 sigmaz

```
std::vector<realtype> sigmaz
```

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

Definition at line 62 of file rdata.h.

10.18.5.6 sz

```
std::vector<realtype> sz
```

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

Definition at line 66 of file rdata.h.

10.18.5.7 ssigmaz

```
std::vector<realtype> ssigmaz
```

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

Definition at line 70 of file rdata.h.

10.18.5.8 rz

```
std::vector<realtype> rz
```

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

Definition at line 73 of file rdata.h.

10.18.5.9 srz

```
std::vector<realtype> srz
```

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

Definition at line 77 of file rdata.h.

10.18.5.10 s2rz

```
std::vector<realtype> s2rz
```

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

Definition at line 81 of file rdata.h.

10.18.5.11 x

```
std::vector<realtype> x
```

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

Definition at line 84 of file rdata.h.

10.18.5.12 sx

```
std::vector<realtype> sx
```

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

Definition at line 88 of file rdata.h.

10.18.5.13 y

```
std::vector<realtype> y
```

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

Definition at line 91 of file rdata.h.

10.18.5.14 sigmay

```
std::vector<realtype> sigmay
```

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

Definition at line 94 of file rdata.h.

10.18.5.15 sy

```
std::vector<realtype> sy
```

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

Definition at line 98 of file rdata.h.

10.18.5.16 ssigmay

```
std::vector<realtype> ssigmay
```

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

Definition at line 102 of file rdata.h.

10.18.5.17 res

```
std::vector<realtype> res
```

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

Definition at line 105 of file rdata.h.

10.18.5.18 sres

```
std::vector<realtype> sres
```

parameter derivative of residual (dimension: nt x ny x nplist, row-major)

Definition at line 109 of file rdata.h.

10.18.5.19 numsteps

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

number of integration steps forward problem (dimension: nt)

Definition at line 112 of file rdata.h.

10.18.5.20 numstepsB

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

number of integration steps backward problem (dimension: nt)

Definition at line 115 of file rdata.h.

10.18.5.21 numrhsevals

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

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

Definition at line 118 of file rdata.h.

10.18.5.22 numrhsevalsB

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

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

Definition at line 121 of file rdata.h.

10.18.5.23 numerrtestfails

```
std::vector<int> numerrtestfails
```

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

Definition at line 124 of file rdata.h.

10.18.5.24 numerrtestfailsB

```
std::vector<int> numerrtestfailsB
```

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

Definition at line 127 of file rdata.h.

10.18.5.25 numnonlinsolvconvfails

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

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

Definition at line 131 of file rdata.h.

10.18.5.26 numnonlinsolvconvfailsB

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

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

Definition at line 135 of file rdata.h.

10.18.5.27 order

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

employed order forward problem (dimension: nt)

Definition at line 138 of file rdata.h.

10.18.5.28 newton_status

```
int newton_status = 0
```

flag indicating success of Newton solver

Definition at line 141 of file rdata.h.

10.18.5.29 newton_time

```
double newton_time = 0.0
```

computation time of the Newton solver [s]

Definition at line 144 of file rdata.h.

10.18.5.30 newton_numsteps

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

number of Newton steps for steady state problem

Definition at line 147 of file rdata.h.

10.18.5.31 newton_numlinsteps

```
std::vector<int> newton_numlinsteps
```

number of linear steps by Newton step for steady state problem

Definition at line 150 of file rdata.h.

10.18.5.32 x0

```
std::vector<realtype> x0
```

preequilibration steady state found be Newton solver

Definition at line 153 of file rdata.h.

10.18.5.33 sx0

```
std::vector<realtype> sx0
```

preequilibration sensitivities found be Newton solver

Definition at line 156 of file rdata.h.

10.18.5.34 llh

```
realtype llh = 0.0
```

likelihood value (double[1])

Definition at line 159 of file rdata.h.

10.18.5.35 chi2

```
realtype chi2 = 0.0
```

chi2 value (double[1])

Definition at line 162 of file rdata.h.

10.18.5.36 sllh

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

parameter derivative of likelihood (dimension: nplist)

Definition at line 165 of file rdata.h.

10.18.5.37 s2llh

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

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

Definition at line 169 of file rdata.h.

10.18.5.38 status

```
int status = 0
```

status code (double[1])

Definition at line 172 of file rdata.h.

10.18.5.39 np

```
const int np
```

total number of model parameters

Definition at line 176 of file rdata.h.

10.18.5.40 nk

```
const int nk
```

number of fixed parameters

Definition at line 178 of file rdata.h.

10.18.5.41 nx

```
const int nx
```

number of states

Definition at line 180 of file rdata.h.

10.18.5.42 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 182 of file rdata.h.

10.18.5.43 ny

```
const int ny
```

number of observables

Definition at line 184 of file rdata.h.

10.18.5.44 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 186 of file rdata.h.

10.18.5.45 nz

```
const int nz
```

number of event outputs

Definition at line 188 of file rdata.h.

10.18.5.46 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 190 of file rdata.h.

10.18.5.47 ne

```
const int ne
```

number of events

Definition at line 192 of file rdata.h.

10.18.5.48 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 194 of file rdata.h.

10.18.5.49 nplist

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 197 of file rdata.h.

10.18.5.50 nmaxevent

```
const int nmaxevent
```

maximal number of occuring events (for every event type)

Definition at line 199 of file rdata.h.

10.18.5.51 nt

```
const int nt
```

number of considered timepoints

Definition at line 201 of file rdata.h.

10.18.5.52 newton_maxsteps

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 203 of file rdata.h.

10.18.5.53 pscale

```
std::vector<AMICI_parameter_scaling> pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 205 of file rdata.h.

10.18.5.54 o2mode

```
const AMICI_o2mode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 207 of file rdata.h.

10.18.5.55 sensi

```
const AMICI_sensi_order sensi
```

sensitivity order

Definition at line 209 of file rdata.h.

10.18.5.56 sensi_meth

```
const AMICI_sensi_meth sensi_meth
```

sensitivity method

Definition at line 211 of file rdata.h.

10.19 SBMLException Class Reference

Inheritance diagram for SBMLException:

10.19.1 Detailed Description

Definition at line 20 of file sbml_import.py.

10.20 SbmlImporter Class Reference

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

Public Member Functions

- def __init__ (self, SBMLFile)

  Create a new Model instance.
- def loadSBMLFile (self, SBMLFile)

  Parse the provided SBML file.
- def sbml2amici (self, modelName, output_dir=None, observables={}, constantParameters=[ ], sigmas={})

  Generate AMICI C++ files for the model provided to the constructor.
- def setName (self, modelName)

  Sets the model name.
- def setPaths (self, output_dir=None)

  Set output paths for the model and create if necessary.
- def processSBML (self, constantParameters=[ ])

  Read parameters, species, reactions, and so on from SBML model.
- def checkSupport (self)

  Check whether all required SBML features are supported.
- def processSpecies (self)

  Get species information from SBML model.
- def processParameters (self, constantParameters=[ ])

  Get parameter information from SBML model.
- def processCompartments (self)

  Get compartment information, stoichiometric matrix and fluxes from SBML model.

- def processReactions (self)

  Get reactions from SBML model.
- def processRules (self)

  Process ∗Rules defined in the SBML model.
- def processVolumeConversion (self)

  Convert equations from amount to volume.
- def processTime (self)

  Convert time_symbol into cpp variable.
- def replaceInAllExpressions (self, old, new)

  Replace 'old' by 'new' in all symbolic expressions.
- def cleanReservedSymbols (self)

  Remove all reserved symbols from self.symbols.
- def replaceSpecialConstants (self)

  Replace all special constants by their respective SBML csymbol definition.
- def getSparseSymbols (self, symbolName)

  Create sparse symbolic matrix.
- def computeModelEquations (self, observables={}, sigmas={})

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

  Perform symbolic computations required for the use of various linear solvers.
- def computeModelEquationsObjectiveFunction (self, observables={}, sigmas={})

  Perform symbolic computations required for objective function evaluation.
- def computeModelEquationsSensitivitesCore (self)

  Perform symbolic computations required for any sensitivity analysis.
- def computeModelEquationsForwardSensitivites (self)

  Perform symbolic computations required for forward sensitivity analysis.
- def computeModelEquationsAdjointSensitivites (self)

  Perform symbolic computations required for adjoint sensitivity analysis.
- def prepareModelFolder (self)

  Remove all files from the model folder.
- def generateCCode (self)

  Create C++ code files for the model based on.
- def compileCCode (self)

  Compile the generated model code.
- def writeIndexFiles (self, name)

  Write index file for a symbolic array.
- def writeFunctionFile (self, function)

  Write the function `function`.
- def getFunctionBody (self, function)

  Generate C++ code for body of function `function`.
- def writeWrapfunctionsCPP (self)

  Write model-specific 'wrapper' file (wrapfunctions.cpp).
- def writeWrapfunctionsHeader (self)

  Write model-specific header file (wrapfunctions.h).
- def writeModelHeader (self)

  Write model-specific header file (MODELNAME.h).
- def writeCMakeFile (self)

  Write CMake CMakeLists.txt file for this model.
- def writeSwigFiles (self)

  Write SWIG interface files for this model.
- def writeModuleSetup (self)

Create a distutils setup.py file for compile the model module.

- def getSymLines (self, symbols, variable, indentLevel)

  Generate C++ code for assigning symbolic terms in symbols to C++ array `variable`.
- def getSparseSymLines (self, symbolList, RowVals, ColPtrs, variable, indentLevel)

  Generate C++ code for assigning sparse symbolic matrix to a C++ array `variable`.
- def printWithException (self, math)

  Generate C++ code for a symbolic expression.

Public Attributes

- Codeprinter

  codeprinter that allows
- functions

  dict carrying function specific definitions
- symbols

  symbolic definitions
- SBMLreader

  the libSBML sbml reader [!not storing this will result in a segfault!]
- sbml_doc

  document carrying the sbml defintion [!not storing this will result in a segfault!]
- sbml

  sbml definition [!not storing this will result in a segfault!]
- modelName

  name of the model that will be used for compilation
- modelPath

  path to the generated model specific files
- modelSwigPath

  path to the generated swig files
- n_species

  number of species
- speciesIndex

  dict that maps species names to indices
- speciesCompartment

  array of compartment for each species
- constantSpecies

  ids of species that are marked as constant
- boundaryConditionSpecies

  ids of species that are marked as boundary condition
- speciesHasOnlySubstanceUnits

  array of flags indicating whether a species has only substance units
- speciesInitial

  array of initial concentrations
- speciesConversionFactor

  array of conversion factors for every
- parameterValues

  array of parameter values
- n_parameters

  number of parameters
- parameterIndex

  dict that maps parameter names to indices

- fixedParameterValues

  array of fixed parameter values
- n_fixed_parameters

  number of fixed parameters
- fixedParameterIndex

  dict that maps fixed parameter names to indices
- compartmentSymbols

  array of compartment ids
- compartmentVolume

  array of compartment volumes
- n_reactions

  number of reactions
- stoichiometricMatrix

  stoichiometrix matrix of the model
- fluxVector

  vector of reaction kinetic laws
- observables

  array of observable definitions
- n_observables

  number of observables

10.20.1 Detailed Description

Definition at line 28 of file sbml_import.py.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 __init__()

```
def __init__ (
            self,
            SBMLFile )
```

Parameters

SBMLFile	Path to SBML file where the model is specified

Returns

SbmlIMporter instance with attached SBML document

Definition at line 131 of file sbml_import.py.

Here is the call graph for this function:

10.20.3 Member Function Documentation

10.20.3.1 loadSBMLFile()

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

Parameters

SBMLFile	path to SBML file

Returns

Definition at line 257 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.2 sbml2amici()

```
def sbml2amici (
              self,
              modelName,
              output_dir = None,
              observables = {},
              constantParameters = [],
              sigmas = {} )
```

Parameters

modelName	name of the model/model directory
output_dir	see sbml_import.setPaths()
observables	dictionary(observableName:formulaString) to be added to the model
sigmas	dictionary(observableName: sigma value or (existing) parameter name)
constantParameters	list of SBML Ids identifying constant parameters

Returns

Definition at line 298 of file sbml_import.py.

Here is the call graph for this function:

10.20.3.3 setName()

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

Parameters

modelName	name of the model (must only contain valid filename characters)

Returns

Definition at line 319 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.4 setPaths()

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

Parameters

output_dir	relative or absolute path where the generated model code is to be placed. will be created if does not exists. defaults to `pwd`/amici-$modelname.

Returns

Definition at line 333 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.5 processSBML()

```
def processSBML (
              self,
              constantParameters = [] )
```

Parameters

constantParameters	list of SBML Ids identifying constant parameters

Returns

Definition at line 356 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.6 checkSupport()

```
def checkSupport (
              self )
```

Returns

Definition at line 378 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.7 processSpecies()

```
def processSpecies (
            self )
```

Returns

Definition at line 409 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.8 processParameters()

```
def processParameters (
            self,
            constantParameters = [] )
```

Parameters

constantParameters	list of SBML Ids identifying constant parameters

Returns

Definition at line 449 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.9 processCompartments()

```
def processCompartments (
              self )
```

Returns

Definition at line 476 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.10 processReactions()

```
def processReactions (
              self )
```

Returns

Definition at line 494 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.11 processRules()

```
def processRules (
              self )
```

Returns

Definition at line 555 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.12 processVolumeConversion()

```
def processVolumeConversion (
              self )
```

Returns

Definition at line 608 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.13 processTime()

```
def processTime (
            self )
```

Returns

Definition at line 626 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.14 replaceInAllExpressions()

```
def replaceInAllExpressions (
            self,
            old,
            new )
```

Parameters

old	symbolic variables to be replaced
new	replacement symbolic variables

Returns

Definition at line 645 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.15 cleanReservedSymbols()

```
def cleanReservedSymbols (
              self )
```

Returns

Definition at line 661 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.16 replaceSpecialConstants()

```
def replaceSpecialConstants (
              self )
```

Returns

Definition at line 680 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.17 getSparseSymbols()

```
def getSparseSymbols (
              self,
              symbolName )
```

Parameters

symbolName	name of the function

Returns

sparseMatrix sparse matrix containing symbolic entries
symbolList symbolic vector containing the list of symbol names
sparseList symbolic vector containing the list of symbol formulas
symbolColPtrs Column pointer as specified in the SlsMat definition in CVODES
symbolRowVals Row Values as specified in the SlsMat definition in CVODES

Definition at line 708 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.18 computeModelEquations()

```
def computeModelEquations (
            self,
            observables = {},
            sigmas = {} )
```

Parameters

observables	dictionary(observableName:formulaString) to be added to the model
sigmas	dictionary(observableName: sigma value or (existing) parameter name)

Returns

Definition at line 741 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.19 computeModelEquationsLinearSolver()

```
def computeModelEquationsLinearSolver (
              self )
```

Returns

Definition at line 763 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.20 computeModelEquationsObjectiveFunction()

```
def computeModelEquationsObjectiveFunction (
              self,
              observables = {},
              sigmas = {} )
```

Parameters

observables	dictionary(observableName:formulaString) to be added to the model
sigmas	dictionary(observableName: sigma value or (existing) parameter name)

Returns

Definition at line 797 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.21 computeModelEquationsSensitivitesCore()

```
def computeModelEquationsSensitivitesCore (
              self )
```

Returns

Definition at line 841 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.22 computeModelEquationsForwardSensitivites()

```
def computeModelEquationsForwardSensitivites (
              self )
```

Returns

Definition at line 870 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.23 computeModelEquationsAdjointSensitivites()

```
def computeModelEquationsAdjointSensitivites (
              self )
```

Returns

Definition at line 885 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.24 prepareModelFolder()

```
def prepareModelFolder (
              self )
```

Returns

Definition at line 910 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.25 generateCCode()

```
def generateCCode (
              self )
```

Returns

Definition at line 927 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.26 compileCCode()

```
def compileCCode (
            self )
```

Returns

Definition at line 954 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.27 writeIndexFiles()

```
def writeIndexFiles (
              self,
              name )
```

Parameters

name	key in self.symbols for which the respective file should be written

Returns

Definition at line 977 of file sbml_import.py.

Here is the caller graph for this function:

10.20.3.28 writeFunctionFile()

```
def writeFunctionFile (
              self,
              function )
```

Parameters

function	name of the function to be written (see self.functions)

Returns

Definition at line 995 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.29 getFunctionBody()

```
def getFunctionBody (
            self,
            function )
```

Parameters

function	name of the function to be written (see self.functions)

Returns

Definition at line 1042 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.30 writeWrapfunctionsCPP()

```
def writeWrapfunctionsCPP (
              self )
```

Returns

Definition at line 1090 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.31 writeWrapfunctionsHeader()

```
def writeWrapfunctionsHeader (
              self )
```

Returns

Definition at line 1105 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.32 writeModelHeader()

```
def writeModelHeader (
              self )
```

Returns

Definition at line 1119 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.33 writeCMakeFile()

```
def writeCMakeFile (
              self )
```

Returns

Definition at line 1152 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.34 writeSwigFiles()

```
def writeSwigFiles (
              self )
```

Returns

Definition at line 1173 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.35 writeModuleSetup()

```
def writeModuleSetup (
              self )
```

Returns

Definition at line 1191 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.36 getSymLines()

```
def getSymLines (
              self,
              symbols,
              variable,
              indentLevel )
```

Parameters

symbols	vectors of symbolic terms
variable	name of the C++ array to assign to
indentLevel	indentation level (number of leading blanks)

Returns

C++ code as list of lines

Definition at line 1218 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.37 getSparseSymLines()

```
def getSparseSymLines (
              self,
              symbolList,
              RowVals,
              ColPtrs,
              variable,
              indentLevel )
```

Parameters

symbolList	vectors of symbolic terms
RowVals	list of row indices of each nonzero entry (see CVODES SlsMat documentation for details)
ColPtrs	list of indices of the first column entries (see CVODES SlsMat documentation for details)
variable	name of the C++ array to assign to
indentLevel	indentation level (number of leading blanks)

Returns

C++ code as list of lines

Definition at line 1245 of file sbml_import.py.

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.38 printWithException()

```
def printWithException (
            self,
            math )
```

Parameters

math	symbolic expression

Returns

C++ code for the specified expression

Exceptions

SBMLException	The specified expression containted an unsupported function

Definition at line 1268 of file sbml_import.py.

Here is the caller graph for this function:

10.21 TemplateAmici Class Reference

Template format used in AMICI (see string.template for more details).

Inheritance diagram for TemplateAmici:

Static Public Attributes

- string delimiter = 'TPL_'

  delimiter that identifies template variables

10.21.1 Detailed Description

Returns

Definition at line 1351 of file sbml_import.py.

10.22 SetupFailure Class Reference

setup failure exception this exception should be thrown when the solver setup failed for this exception we can assume that we cannot recover from the exception and an error will be thrown

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

Inheritance diagram for SetupFailure:

Public Member Functions

- SetupFailure (const char *msg)

10.22.1 Detailed Description

Definition at line 166 of file exception.h.

10.22.2 Constructor & Destructor Documentation

10.22.2.1 SetupFailure()

```
SetupFailure (
            const char * msg )
```

constructor, simply calls AmiException constructor

Parameters

in	*msg*	

Definition at line 171 of file exception.h.

10.23 Solver Class Reference

```
#include <solver.h>
```

Public Member Functions

- Solver (const Solver &other)

  Solver copy constructor.
- virtual Solver ∗ clone () const =0

  Clone this instance.
- void setupAMI (ForwardProblem ∗fwd, Model ∗model)

  setupAMIs initialises the ami memory object
- void setupAMIB (BackwardProblem ∗bwd, Model ∗model)
- virtual void AMIGetSens (realtype ∗tret, AmiVectorArray ∗yySout)=0
- void getDiagnosis (const int it, ReturnData ∗rdata)
- void getDiagnosisB (const int it, ReturnData ∗rdata, const BackwardProblem ∗bwd)
- virtual void AMIGetRootInfo (int ∗rootsfound)=0
- virtual void AMIReInit (realtype t0, AmiVector ∗yy0, AmiVector ∗yp0)=0
- virtual void AMISensReInit (int ism, AmiVectorArray ∗yS0, AmiVectorArray ∗ypS0)=0
- virtual void AMICalcIC (realtype tout1, AmiVector ∗x, AmiVector ∗dx)=0
- virtual void AMICalcICB (int which, realtype tout1, AmiVector ∗xB, AmiVector ∗dxB)=0
- virtual int AMISolve (realtype tout, AmiVector ∗yret, AmiVector ∗ypret, realtype ∗tret, int itask)=0
- virtual int AMISolveF (realtype tout, AmiVector ∗yret, AmiVector ∗ypret, realtype ∗tret, int itask, int ∗ncheck↩ Ptr)=0
- virtual void AMISolveB (realtype tBout, int itaskB)=0
- virtual void AMISetStopTime (realtype tstop)=0
- virtual void AMIReInitB (int which, realtype tB0, AmiVector ∗yyB0, AmiVector ∗ypB0)=0
- virtual void AMIGetB (int which, realtype ∗tret, AmiVector ∗yy, AmiVector ∗yp)=0
- virtual void AMIGetQuadB (int which, realtype ∗tret, AmiVector ∗qB)=0
- virtual void AMIQuadReInitB (int which, AmiVector ∗yQB0)=0
- virtual void turnOffRootFinding ()=0
- AMICI_sensi_meth getSensitivityMethod () const
- void setSensitivityMethod (AMICI_sensi_meth sensi_meth)

  setSensitivityMethod
- int getNewtonMaxSteps () const

  getNewtonMaxSteps
- void setNewtonMaxSteps (int newton_maxsteps)

  setNewtonMaxSteps
- bool getNewtonPreequilibration () const

  getNewtonPreequilibration
- void setNewtonPreequilibration (bool newton_preeq)

  setNewtonPreequilibration
- int getNewtonMaxLinearSteps () const

  getNewtonMaxLinearSteps
- void setNewtonMaxLinearSteps (int newton_maxlinsteps)

  setNewtonMaxLinearSteps
- AMICI_sensi_order getSensitivityOrder () const

  getSensitivityOrder
- void setSensitivityOrder (AMICI_sensi_order sensi)

  setSensitivityOrder
- double getRelativeTolerance () const

  getRelativeTolerance
- void setRelativeTolerance (double rtol)

  setRelativeTolerance
- double getAbsoluteTolerance () const

  getAbsoluteTolerance

- void setAbsoluteTolerance (double atol)

  setAbsoluteTolerance
- double getRelativeToleranceQuadratures () const

  getRelativeToleranceQuadratures
- void setRelativeToleranceQuadratures (double rtol)

  setRelativeToleranceQuadratures
- double getAbsoluteToleranceQuadratures () const

  getAbsoluteToleranceQuadratures
- void setAbsoluteToleranceQuadratures (double atol)

  setAbsoluteToleranceQuadratures
- int getMaxSteps () const

  getMaxSteps
- void setMaxSteps (int maxsteps)

  setMaxSteps
- int getMaxStepsBackwardProblem () const

  getMaxStepsBackwardProblem
- void setMaxStepsBackwardProblem (int maxsteps)

  setMaxStepsBackwardProblem
- LinearMultistepMethod getLinearMultistepMethod () const

  getLinearMultistepMethod
- void setLinearMultistepMethod (LinearMultistepMethod lmm)

  setLinearMultistepMethod
- NonlinearSolverIteration getNonlinearSolverIteration () const

  getNonlinearSolverIteration
- void setNonlinearSolverIteration (NonlinearSolverIteration iter)

  setNonlinearSolverIteration
- InterpolationType getInterpolationType () const

  getInterpolationType
- void setInterpolationType (InterpolationType interpType)

  setInterpolationType
- StateOrdering getStateOrdering () const

  getStateOrdering
- void setStateOrdering (StateOrdering ordering)

  setStateOrdering
- int getStabilityLimitFlag () const

  getStabilityLimitFlag
- void setStabilityLimitFlag (int stldet)

  setStabilityLimitFlag
- LinearSolver getLinearSolver () const

  getLinearSolver
- void setLinearSolver (LinearSolver linsol)

  setLinearSolver
- InternalSensitivityMethod getInternalSensitivityMethod () const

  getInternalSensitivityMethod
- void setInternalSensitivityMethod (InternalSensitivityMethod ism)

  setInternalSensitivityMethod

Protected Member Functions

- virtual void init (AmiVector ∗x, AmiVector ∗dx, realtype t)=0
- virtual void binit (int which, AmiVector ∗xB, AmiVector ∗dxB, realtype t)=0
- virtual void qbinit (int which, AmiVector ∗qBdot)=0
- virtual void rootInit (int ne)=0
- virtual void sensInit1 (AmiVectorArray ∗sx, AmiVectorArray ∗sdx, int nplist)=0
- virtual void setDenseJacFn ()=0
- virtual void setSparseJacFn ()=0
- virtual void setBandJacFn ()=0
- virtual void setJacTimesVecFn ()=0
- virtual void setDenseJacFnB (int which)=0
- virtual void setSparseJacFnB (int which)=0
- virtual void setBandJacFnB (int which)=0
- virtual void setJacTimesVecFnB (int which)=0
- virtual void ∗ AMICreate (int lmm, int iter)=0
- virtual void AMISStolerances (double rtol, double atol)=0
- virtual void AMISensSStolerances (double rtol, double ∗atol)=0
- virtual void AMISetSensErrCon (bool error_corr)=0
- virtual void AMISetQuadErrConB (int which, bool flag)=0
- virtual void AMISetErrHandlerFn ()=0
- virtual void AMISetUserData (Model ∗model)=0
- virtual void AMISetUserDataB (int which, Model ∗model)=0
- virtual void AMISetMaxNumSteps (long int mxsteps)=0
- virtual void AMISetMaxNumStepsB (int which, long int mxstepsB)=0
- virtual void AMISetStabLimDet (int stldet)=0
- virtual void AMISetStabLimDetB (int which, int stldet)=0
- virtual void AMISetId (Model ∗model)=0
- virtual void AMISetSuppressAlg (bool flag)=0
- virtual void AMISetSensParams (realtype ∗p, realtype ∗pbar, int ∗plist)=0
- virtual void AMIGetDky (realtype t, int k, AmiVector ∗dky)=0
- virtual void AMIFree ()=0
- virtual void AMIAdjInit (long int steps, int interp)=0
- virtual void AMICreateB (int lmm, int iter, int ∗which)=0
- virtual void AMISStolerancesB (int which, realtype relTolB, realtype absTolB)=0
- virtual void AMIQuadSStolerancesB (int which, realtype reltolQB, realtype abstolQB)=0
- virtual void AMIDense (int nx)=0
- virtual void AMIDenseB (int which, int nx)=0
- virtual void AMIBand (int nx, int ubw, int lbw)=0
- virtual void AMIBandB (int which, int nx, int ubw, int lbw)=0
- virtual void AMIDiag ()=0
- virtual void AMIDiagB (int which)=0
- virtual void AMISpgmr (int prectype, int maxl)=0
- virtual void AMISpgmrB (int which, int prectype, int maxl)=0
- virtual void AMISpbcg (int prectype, int maxl)=0
- virtual void AMISpbcgB (int which, int prectype, int maxl)=0
- virtual void AMISptfqmr (int prectype, int maxl)=0
- virtual void AMISptfqmrB (int which, int prectype, int maxl)=0
- virtual void AMIKLU (int nx, int nnz, int sparsetype)=0
- virtual void AMIKLUSetOrdering (int ordering)=0
- virtual void AMIKLUSetOrderingB (int which, int ordering)=0
- virtual void AMIKLUB (int which, int nx, int nnz, int sparsetype)=0
- virtual void AMIGetNumSteps (void ∗ami_mem, long int ∗numsteps)=0
- virtual void AMIGetNumRhsEvals (void ∗ami_mem, long int ∗numrhsevals)=0
- virtual void AMIGetNumErrTestFails (void ∗ami_mem, long int ∗numerrtestfails)=0

- virtual void AMIGetNumNonlinSolvConvFails (void ∗ami_mem, long int ∗numnonlinsolvconvfails)=0
- virtual void AMIGetLastOrder (void ∗ami_mem, int ∗order)=0
- virtual void ∗ AMIGetAdjBmem (void ∗ami_mem, int which)=0
- void initializeLinearSolver (Model ∗model)
- void initializeLinearSolverB (Model ∗model, int which)

Static Protected Member Functions

- static void wrapErrHandlerFn (int error_code, const char ∗module, const char ∗function, char ∗msg, void ∗eh_data)

Protected Attributes

- void ∗ ami_mem = nullptr
- bool solverWasCalled = false

Friends

- template<class Archive >
  void boost::serialization::serialize (Archive &ar, Solver &r, const unsigned int version)

  Serialize Solver (see boost::serialization::serialize)
- bool operator== (const Solver &a, const Solver &b)

  Check equality of data members.

10.23.1 Detailed Description

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

Definition at line 38 of file solver.h.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 Solver()

```
Solver (
            const Solver & other )
```

Parameters

other	

Definition at line 46 of file solver.h.

10.23.3 Member Function Documentation

10.23.3.1 clone()

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

Returns

The clone

10.23.3.2 setupAMI()

```
void setupAMI (
            ForwardProblem * fwd,
            Model * model )
```

Parameters

fwd	pointer to forward problem
model	pointer to the model object

Definition at line 23 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.23.3.3 setupAMIB()

```
void setupAMIB (
            BackwardProblem * bwd,
            Model * model )
```

setupAMIB initialises the AMI memory object for the backwards problem

Parameters

bwd	pointer to backward problem
model	pointer to the model object

Definition at line 113 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.23.3.4 AMIGetSens()

```
virtual void AMIGetSens (
            realtype * tret,
            AmiVectorArray * yySout )  [pure virtual]
```

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

Parameters

tret	time at which the sensitivities should be computed
yySout	vector with sensitivities

10.23.3.5 getDiagnosis()

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

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

Parameters

it	time-point index
rdata	pointer to the return data object

Definition at line 215 of file solver.cpp.

Here is the call graph for this function:

10.23.3.6 getDiagnosisB()

```
void getDiagnosisB (
            const int it,
            ReturnData * rdata,
            const BackwardProblem * bwd )
```

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

Parameters

it	time-point index
rdata	pointer to the return data object
bwd	pointer to backward problem

Definition at line 245 of file solver.cpp.

Here is the call graph for this function:

10.23.3.7 AMIGetRootInfo()

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

AMIGetRootInfo extracts information which event occured

Parameters

rootsfound	array with flags indicating whether the respective event occured

10.23.3.8 AMIReInit()

```
virtual void AMIReInit (
            realtype t0,
            AmiVector * yy0,
            AmiVector * yp0 )  [pure virtual]
```

AMIReInit reinitializes the states in the solver after an event occurence

Parameters

t0	new timepoint
yy0	new state variables
yp0	new derivative state variables (DAE only)

10.23.3.9 AMISensReInit()

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

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

Parameters

ism	sensitivity mode
yS0	new state sensitivity
ypS0	new derivative state sensitivities (DAE only)

10.23.3.10 AMICalcIC()

```
virtual void AMICalcIC (
            realtype tout1,
            AmiVector * x,
            AmiVector * dx )  [pure virtual]
```

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

Parameters

tout1	next timepoint to be computed (sets timescale)
x	initial state variables
dx	initial derivative state variables (DAE only)

Here is the caller graph for this function:

10.23.3.11 AMICalcICB()

```
virtual void AMICalcICB (
            int which,
            realtype tout1,
            AmiVector * xB,
            AmiVector * dxB )  [pure virtual]
```

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

Parameters

which	identifier of the backwards problem
tout1	next timepoint to be computed (sets timescale)
xB	states of final solution of the forward problem
dxB	derivative states of final solution of the forward problem (DAE only)

Here is the caller graph for this function:

10.23.3.12 AMISolve()

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

AMISolve solves the forward problem until a predefined timepoint

Parameters

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

Returns

status flag indicating success of execution

Here is the caller graph for this function:

10.23.3.13 AMISolveF()

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

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

Parameters

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

Returns

status flag indicating success of execution

Here is the caller graph for this function:

10.23.3.14 AMISolveB()

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

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

Parameters

tBout	timepoint until which simulation should be performed
itaskB	task identifier, can be CV_NORMAL or CV_ONE_STEP

Here is the caller graph for this function:

10.23.3.15 AMISetStopTime()

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

AMISetStopTime sets a timepoint at which the simulation will be stopped

Parameters

tstop	timepoint until which simulation should be performed

Here is the caller graph for this function:

10.23.3.16 AMIReInitB()

```
virtual void AMIReInitB (
            int which,
            realtype tB0,
            AmiVector * yyB0,
            AmiVector * ypB0 )  [pure virtual]
```

AMIReInitB reinitializes the adjoint states after an event occurence

Parameters

which	identifier of the backwards problem
tB0	new timepoint
yyB0	new adjoint state variables
ypB0	new adjoint derivative state variables (DAE only)

Here is the caller graph for this function:

10.23.3.17 AMIGetB()

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

AMIGetB returns the current adjoint states

Parameters

which	identifier of the backwards problem
tret	time at which the adjoint states should be computed
yy	adjoint state variables
yp	adjoint derivative state variables (DAE only)

Here is the caller graph for this function:

10.23.3.18 AMIGetQuadB()

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

AMIGetQuadB returns the current adjoint states

Parameters

which	identifier of the backwards problem
tret	time at which the adjoint states should be computed
qB	adjoint quadrature state variables

Here is the caller graph for this function:

10.23.3.19 AMIQuadReInitB()

```
virtual void AMIQuadReInitB (
            int which,
            AmiVector * yQB0 )  [pure virtual]
```

AMIReInitB reinitializes the adjoint states after an event occurence

Parameters

which	identifier of the backwards problem
yQB0	new adjoint quadrature state variables

Here is the caller graph for this function:

10.23.3.20 turnOffRootFinding()

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

turnOffRootFinding disables rootfinding Here is the caller graph for this function:

10.23.3.21 getSensitivityMethod()

```
AMICI_sensi_meth getSensitivityMethod ( ) const
```

sensitivity method

Returns

method enum

Definition at line 243 of file solver.h.

10.23.3.22 setSensitivityMethod()

```
void setSensitivityMethod (
            AMICI_sensi_meth sensi_meth )
```

Parameters

sensi_meth	

Definition at line 251 of file solver.h.

10.23.3.23 getNewtonMaxSteps()

```
int getNewtonMaxSteps ( ) const
```

Returns

Definition at line 259 of file solver.h.

10.23.3.24 setNewtonMaxSteps()

```
void setNewtonMaxSteps (
            int newton_maxsteps )
```

Parameters

newton_maxsteps	

Definition at line 267 of file solver.h.

10.23.3.25 getNewtonPreequilibration()

```
bool getNewtonPreequilibration ( ) const
```

Returns

Definition at line 275 of file solver.h.

Here is the caller graph for this function:

10.23.3.26 setNewtonPreequilibration()

```
void setNewtonPreequilibration (
            bool newton_preeq )
```

Parameters

newton_preeq	

Definition at line 283 of file solver.h.

10.23.3.27 getNewtonMaxLinearSteps()

```
int getNewtonMaxLinearSteps ( ) const
```

Returns

Definition at line 292 of file solver.h.

10.23.3.28 setNewtonMaxLinearSteps()

```
void setNewtonMaxLinearSteps (
            int newton_maxlinsteps )
```

Parameters

newton_maxlinsteps	

Definition at line 300 of file solver.h.

10.23.3.29 getSensitivityOrder()

```
AMICI_sensi_order getSensitivityOrder ( ) const
```

Returns

Definition at line 308 of file solver.h.

10.23.3.30 setSensitivityOrder()

```
void setSensitivityOrder (
            AMICI_sensi_order sensi )
```

Parameters

sensi	

Definition at line 316 of file solver.h.

10.23.3.31 getRelativeTolerance()

```
double getRelativeTolerance ( ) const
```

Returns

Definition at line 333 of file solver.h.

10.23.3.32 setRelativeTolerance()

```
void setRelativeTolerance (
            double rtol )
```

Parameters

rtol	

Definition at line 341 of file solver.h.

10.23.3.33 getAbsoluteTolerance()

```
double getAbsoluteTolerance ( ) const
```

Returns

Definition at line 349 of file solver.h.

10.23.3.34 setAbsoluteTolerance()

```
void setAbsoluteTolerance (
            double atol )
```

Parameters

atol	

Definition at line 357 of file solver.h.

10.23.3.35 getRelativeToleranceQuadratures()

```
double getRelativeToleranceQuadratures ( ) const
```

Returns

Definition at line 365 of file solver.h.

10.23.3.36 setRelativeToleranceQuadratures()

```
void setRelativeToleranceQuadratures (
            double rtol )
```

Parameters

rtol	

Definition at line 373 of file solver.h.

10.23.3.37 getAbsoluteToleranceQuadratures()

```
double getAbsoluteToleranceQuadratures ( ) const
```

Returns

Definition at line 381 of file solver.h.

10.23.3.38 setAbsoluteToleranceQuadratures()

```
void setAbsoluteToleranceQuadratures (
            double atol )
```

Parameters

atol	

Definition at line 389 of file solver.h.

10.23.3.39 getMaxSteps()

```
int getMaxSteps ( ) const
```

Returns

Definition at line 397 of file solver.h.

10.23.3.40 setMaxSteps()

```
void setMaxSteps (
            int maxsteps )
```

Parameters

maxsteps	

Definition at line 405 of file solver.h.

10.23.3.41 getMaxStepsBackwardProblem()

```
int getMaxStepsBackwardProblem ( ) const
```

Returns

Definition at line 413 of file solver.h.

10.23.3.42 setMaxStepsBackwardProblem()

```
void setMaxStepsBackwardProblem (
            int maxsteps )
```

Parameters

maxsteps	

Definition at line 421 of file solver.h.

10.23.3.43 getLinearMultistepMethod()

```
LinearMultistepMethod getLinearMultistepMethod ( ) const
```

Returns

Definition at line 429 of file solver.h.

10.23.3.44 setLinearMultistepMethod()

```
void setLinearMultistepMethod (
            LinearMultistepMethod lmm )
```

Parameters

lmm	

Definition at line 437 of file solver.h.

10.23.3.45 getNonlinearSolverIteration()

```
NonlinearSolverIteration getNonlinearSolverIteration ( ) const
```

Returns

Definition at line 448 of file solver.h.

10.23.3.46 setNonlinearSolverIteration()

```
void setNonlinearSolverIteration (
            NonlinearSolverIteration iter )
```

Parameters

iter	

Definition at line 456 of file solver.h.

10.23.3.47 getInterpolationType()

```
InterpolationType getInterpolationType ( ) const
```

Returns

Definition at line 467 of file solver.h.

10.23.3.48 setInterpolationType()

```
void setInterpolationType (
            InterpolationType interpType )
```

Parameters

interpType	

Definition at line 475 of file solver.h.

10.23.3.49 getStateOrdering()

```
StateOrdering getStateOrdering ( ) const
```

Returns

Definition at line 483 of file solver.h.

Here is the caller graph for this function:

10.23.3.50 setStateOrdering()

```
void setStateOrdering (
            StateOrdering ordering )
```

Parameters

ordering	

Definition at line 491 of file solver.h.

10.23.3.51 getStabilityLimitFlag()

```
int getStabilityLimitFlag ( ) const
```

Returns

Definition at line 499 of file solver.h.

10.23.3.52 setStabilityLimitFlag()

```
void setStabilityLimitFlag (
            int stldet )
```

Parameters

stldet	

Definition at line 507 of file solver.h.

10.23.3.53 getLinearSolver()

```
LinearSolver getLinearSolver ( ) const
```

Returns

Definition at line 515 of file solver.h.

10.23.3.54 setLinearSolver()

```
void setLinearSolver (
            LinearSolver linsol )
```

Parameters

linsol	

Definition at line 523 of file solver.h.

10.23.3.55 getInternalSensitivityMethod()

```
InternalSensitivityMethod getInternalSensitivityMethod ( ) const
```

Returns

Definition at line 531 of file solver.h.

10.23.3.56 setInternalSensitivityMethod()

```
void setInternalSensitivityMethod (
            InternalSensitivityMethod ism )
```

Parameters

ism	

Definition at line 539 of file solver.h.

10.23.3.57 init()

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

init initialises the states at the specified initial timepoint

Parameters

x	initial state variables
dx	initial derivative state variables (DAE only)
t	initial timepoint

Here is the caller graph for this function:

10.23.3.58 binit()

```
virtual void binit (
            int which,
            AmiVector * xB,
            AmiVector * dxB,
            realtype t )  [protected], [pure virtual]
```

binit initialises the adjoint states at the specified final timepoint

Parameters

which	identifier of the backwards problem
xB	initial adjoint state variables
dxB	initial adjoint derivative state variables (DAE only)
t	final timepoint

Here is the caller graph for this function:

10.23.3.59 qbinit()

```
virtual void qbinit (
            int which,
            AmiVector * qBdot )  [protected], [pure virtual]
```

qbinit initialises the quadrature states at the specified final timepoint

Parameters

which	identifier of the backwards problem
qBdot	initial adjoint quadrature state variables

Here is the caller graph for this function:

10.23.3.60 rootInit()

```
virtual void rootInit (
            int ne )  [protected], [pure virtual]
```

RootInit initialises the rootfinding for events

Parameters

ne	number of different events

Here is the caller graph for this function:

10.23.3.61 sensInit1()

```
virtual void sensInit1 (
            AmiVectorArray * sx,
            AmiVectorArray * sdx,
            int nplist )  [protected], [pure virtual]
```

SensInit1 initialises the sensitivities at the specified initial timepoint

Parameters

sx	initial state sensitivities
sdx	initial derivative state sensitivities (DAE only)
nplist	number parameter wrt which sensitivities are to be computed

Here is the caller graph for this function:

10.23.3.62 setDenseJacFn()

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

SetDenseJacFn sets the dense Jacobian function Here is the caller graph for this function:

10.23.3.63 setSparseJacFn()

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

SetSparseJacFn sets the sparse Jacobian function Here is the caller graph for this function:

10.23.3.64 setBandJacFn()

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

SetBandJacFn sets the banded Jacobian function Here is the caller graph for this function:

10.23.3.65 setJacTimesVecFn()

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

SetJacTimesVecFn sets the Jacobian vector multiplication function Here is the caller graph for this function:

10.23.3.66 setDenseJacFnB()

```
virtual void setDenseJacFnB (
            int which )  [protected], [pure virtual]
```

SetDenseJacFn sets the dense Jacobian function

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

10.23.3.67 setSparseJacFnB()

```
virtual void setSparseJacFnB (
            int which )  [protected], [pure virtual]
```

SetSparseJacFn sets the sparse Jacobian function

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

10.23.3.68 setBandJacFnB()

```
virtual void setBandJacFnB (
            int which )  [protected], [pure virtual]
```

SetBandJacFn sets the banded Jacobian function

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

10.23.3.69 setJacTimesVecFnB()

```
virtual void setJacTimesVecFnB (
            int which )  [protected], [pure virtual]
```

SetJacTimesVecFn sets the Jacobian vector multiplication function

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

10.23.3.70 wrapErrHandlerFn()

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

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

Parameters

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

Definition at line 173 of file solver.cpp.

10.23.3.71 AMICreate()

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

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

Parameters

lmm	linear multistep method CV_ADAMS or CV_BDF
iter	nonlinear solver method CV_NEWTON or CV_FUNCTIONAL

Here is the caller graph for this function:

10.23.3.72 AMISStolerances()

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

AMISStolerances sets scalar relative and absolute tolerances for the forward problem

Parameters

rtol	relative tolerances
atol	absolute tolerances

Here is the caller graph for this function:

10.23.3.73 AMISensSStolerances()

```
virtual void AMISensSStolerances (
            double rtol,
            double * atol )  [protected], [pure virtual]
```

AMISensSStolerances activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

rtol	relative tolerances
atol	array of absolute tolerances for every sensitivy variable

Here is the caller graph for this function:

10.23.3.74 AMISetSensErrCon()

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

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

Parameters

error_corr	activation flag

Here is the caller graph for this function:

10.23.3.75 AMISetQuadErrConB()

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

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

Parameters

which	identifier of the backwards problem
flag	activation flag

Here is the caller graph for this function:

10.23.3.76 AMISetErrHandlerFn()

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

AMISetErrHandlerFn attaches the error handler function (errMsgIdAndTxt) to the solver Here is the caller graph for this function:

10.23.3.77 AMISetUserData()

```
virtual void AMISetUserData (
            Model * model ) [protected], [pure virtual]
```

AMISetUserData attaches the user data instance (here this is a Model) to the forward problem

Parameters

model	Model instance,

Here is the caller graph for this function:

10.23.3.78 AMISetUserDataB()

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

AMISetUserDataB attaches the user data instance (here this is a Model) to the backward problem

Parameters

which	identifier of the backwards problem
model	Model instance,

Here is the caller graph for this function:

10.23.3.79 AMISetMaxNumSteps()

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

AMISetMaxNumSteps specifies the maximum number of steps for the forward problem

Parameters

mxsteps	number of steps

Here is the caller graph for this function:

10.23.3.80 AMISetMaxNumStepsB()

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

AMISetMaxNumStepsB specifies the maximum number of steps for the forward problem

Parameters

which	identifier of the backwards problem
mxstepsB	number of steps

Here is the caller graph for this function:

10.23.3.81 AMISetStabLimDet()

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

AMISetStabLimDet activates stability limit detection for the forward problem

Parameters

stldet	flag for stability limit detection (TRUE or FALSE)

Here is the caller graph for this function:

10.23.3.82 AMISetStabLimDetB()

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

AMISetStabLimDetB activates stability limit detection for the backward problem

Parameters

which	identifier of the backwards problem
stldet	flag for stability limit detection (TRUE or FALSE)

Here is the caller graph for this function:

10.23.3.83 AMISetId()

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

AMISetId specify algebraic/differential components (DAE only)

Parameters

model	model specification

Here is the caller graph for this function:

10.23.3.84 AMISetSuppressAlg()

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

AMISetId deactivates error control for algebraic components (DAE only)

Parameters

flag	deactivation flag

Here is the caller graph for this function:

10.23.3.85 AMISetSensParams()

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

AMISetSensParams specifies the scaling and indexes for sensitivity computation

Parameters

p	paramaters
pbar	parameter scaling constants
plist	parameter index list

Here is the caller graph for this function:

10.23.3.86 AMIGetDky()

```
virtual void AMIGetDky (
            realtype t,
            int k,
            AmiVector * dky )  [protected], [pure virtual]
```

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

Parameters

t	timepoint
k	derivative order
dky	interpolated solution

10.23.3.87 AMIFree()

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

AMIFree frees allocation solver memory Here is the caller graph for this function:

10.23.3.88 AMIAdjInit()

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

AMIAdjInit initializes the adjoint problem

Parameters

steps	number of integration points between checkpoints
interp	interpolation type, can be CV_POLYNOMIAL or CV_HERMITE

Here is the caller graph for this function:

10.23.3.89 AMICreateB()

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

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

Parameters

which	identifier of the backwards problem
lmm	linear multistep method CV_ADAMS or CV_BDF
iter	nonlinear solver method CV_NEWTON or CV_FUNCTIONAL

Here is the caller graph for this function:

10.23.3.90 AMISStolerancesB()

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

AMISStolerancesB sets relative and absolute tolerances for the backward problem

Parameters

which	identifier of the backwards problem
relTolB	relative tolerances
absTolB	absolute tolerances

Here is the caller graph for this function:

10.23.3.91 AMIQuadSStolerancesB()

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

AMISStolerancesB sets relative and absolute tolerances for the quadrature backward problem

Parameters

which	identifier of the backwards problem
reltolQB	relative tolerances
abstolQB	absolute tolerances

Here is the caller graph for this function:

10.23.3.92 AMIDense()

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

AMIDense attaches a dense linear solver to the forward problem

Parameters

nx	number of state variables

Here is the caller graph for this function:

10.23.3.93 AMIDenseB()

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

AMIDenseB attaches a dense linear solver to the backward problem

Parameters

which	identifier of the backwards problem
nx	number of state variables

Here is the caller graph for this function:

10.23.3.94 AMIBand()

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

AMIBand attaches a banded linear solver to the forward problem

Parameters

nx	number of state variables
ubw	upper matrix bandwidth
lbw	lower matrix bandwidth

Here is the caller graph for this function:

10.23.3.95 AMIBandB()

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

AMIBandB attaches a banded linear solver to the backward problem

Parameters

which	identifier of the backwards problem
nx	number of state variables
ubw	upper matrix bandwidth
lbw	lower matrix bandwidth

Here is the caller graph for this function:

10.23.3.96 AMIDiag()

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

AMIDiag attaches a diagonal linear solver to the forward problem Here is the caller graph for this function:

10.23.3.97 AMIDiagB()

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

AMIDiagB attaches a diagonal linear solver to the backward problem

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

10.23.3.98 AMISpgmr()

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

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

Parameters

prectype	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
maxl	maximum Kryloc subspace dimension

Here is the caller graph for this function:

10.23.3.99 AMISpgmrB()

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

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

Parameters

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

Here is the caller graph for this function:

10.23.3.100 AMISpbcg()

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

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

Parameters

prectype	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
maxl	maximum Kryloc subspace dimension

Here is the caller graph for this function:

10.23.3.101 AMISpbcgB()

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

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

Parameters

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

Here is the caller graph for this function:

10.23.3.102 AMISptfqmr()

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

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

Parameters

prectype	preconditioner type PREC_NONE, PREC_LEFT, PREC_RIGHT or PREC_BOTH
maxl	maximum Kryloc subspace dimension

Here is the caller graph for this function:

10.23.3.103 AMISptfqmrB()

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

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

Parameters

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

Here is the caller graph for this function:

10.23.3.104 AMIKLU()

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

AMIKLU attaches a sparse linear solver to the forward problem

Parameters

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

Here is the caller graph for this function:

10.23.3.105 AMIKLUSetOrdering()

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

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

Parameters

ordering	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering

Here is the caller graph for this function:

10.23.3.106 AMIKLUSetOrderingB()

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

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

Parameters

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

Here is the caller graph for this function:

10.23.3.107 AMIKLUB()

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

AMIKLUB attaches a sparse linear solver to the forward problem

Parameters

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

Here is the caller graph for this function:

10.23.3.108 AMIGetNumSteps()

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

AMIGetNumSteps reports the number of solver steps

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numsteps	output array

Here is the caller graph for this function:

10.23.3.109 AMIGetNumRhsEvals()

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

AMIGetNumRhsEvals reports the number of right hand evaluations

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numrhsevals	output array

Here is the caller graph for this function:

10.23.3.110 AMIGetNumErrTestFails()

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

AMIGetNumErrTestFails reports the number of local error test failures

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numerrtestfails	output array

Here is the caller graph for this function:

10.23.3.111 AMIGetNumNonlinSolvConvFails()

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

AMIGetNumNonlinSolvConvFails reports the number of nonlinear convergence failures

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
numnonlinsolvconvfails	output array

Here is the caller graph for this function:

10.23.3.112 AMIGetLastOrder()

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

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

Parameters

ami_mem	pointer to the solver memory instance (can be from forward or backward problem)
order	output array

Here is the caller graph for this function:

10.23.3.113 AMIGetAdjBmem()

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

AMIGetAdjBmem retrieves the solver memory instance for the backward problem

Parameters

which	identifier of the backwards problem
ami_mem	pointer to the forward solver memory instance

Returns

ami_memB pointer to the backward solver memory instance

Here is the caller graph for this function:

10.23.3.114 initializeLinearSolver()

```
void initializeLinearSolver (
            Model * model )  [protected]
```

setLinearSolver sets the linear solver for the forward problem

Parameters

model	pointer to the model object

Definition at line 270 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.23.3.115 initializeLinearSolverB()

```
void initializeLinearSolverB (
            Model * model,
            int which )  [protected]
```

Sets the linear solver for the backward problem

Parameters

model	pointer to the model object
which	index of the backward problem

Definition at line 347 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.23.4 Friends And Related Function Documentation

10.23.4.1 boost::serialization::serialize

```
void boost::serialization::serialize (
            Archive & ar,
            Solver & r,
            const unsigned int version )  [friend]
```

Parameters

ar	Archive to serialize to
r	Data to serialize
version	Version number

10.23.4.2 operator==

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

Parameters

a	
b	

Returns

Definition at line 424 of file solver.cpp.

10.23.5 Member Data Documentation

10.23.5.1 ami_mem

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

pointer to ami memory block

Definition at line 1058 of file solver.h.

10.23.5.2 solverWasCalled

```
bool solverWasCalled = false  [protected]
```

flag indicating whether the solver was called

Definition at line 1061 of file solver.h.

10.24 SteadystateProblem Class Reference

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

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

Public Member Functions

- void workSteadyStateProblem (ReturnData *rdata, Solver *solver, Model *model, int it)
- void applyNewtonsMethod (ReturnData *rdata, Model *model, NewtonSolver *newtonSolver, int newton_try)
- void getNewtonOutput (ReturnData *rdata, Model *model, int newton_status, double run_time, int it)
- void getNewtonSimulation (ReturnData *rdata, Solver *solver, Model *model, int it)
- SteadystateProblem (realtype *t, AmiVector *x, AmiVectorArray *sx)

10.24.1 Detailed Description

Definition at line 22 of file steadystateproblem.h.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 SteadystateProblem()

```
SteadystateProblem (
            realtype * t,
            AmiVector * x,
            AmiVectorArray * sx )
```

default constructor

Parameters

t	pointer to time variable
x	pointer to state variables
sx	pointer to state sensitivity variables

Definition at line 47 of file steadystateproblem.h.

10.24.3 Member Function Documentation

10.24.3.1 workSteadyStateProblem()

```
void workSteadyStateProblem (
            ReturnData * rdata,
            Solver * solver,
            Model * model,
            int it )
```

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

Parameters

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

Definition at line 17 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.24.3.2 applyNewtonsMethod()

```
void applyNewtonsMethod (
            ReturnData * rdata,
            Model * model,
            NewtonSolver * newtonSolver,
            int newton_try )
```

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

Parameters

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

Definition at line 93 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.24.3.3 getNewtonOutput()

```
void getNewtonOutput (
            ReturnData * rdata,
            Model * model,
            int newton_status,
            double run_time,
            int it )
```

Stores output of workSteadyStateProblem in return data

Parameters

in	*model*	pointer to the AMICI model object **Type**: Model
in	*newton_status*	integer flag indicating when a steady state was found
in	*run_time*	double coputation time of the solver in milliseconds
out	*rdata*	pointer to the return data object **Type**: ReturnData
in	*it*	current timepoint index, <0 indicates preequilibration **Type**: int

Definition at line 196 of file steadystateproblem.cpp.

Here is the caller graph for this function:

10.24.3.4 getNewtonSimulation()

```
void getNewtonSimulation (
            ReturnData * rdata,
```

```
Solver * solver,
Model * model,
int it )
```

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

Parameters

in	*solver*	pointer to the AMICI solver object **Type**: Solver
in	*model*	pointer to the AMICI model object **Type**: Model
out	*rdata*	pointer to the return data object **Type**: ReturnData
in	*it*	current timepoint index, <0 indicates preequilibration **Type**: int

Definition at line 234 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.25 amidata Class Reference

AMIDATA provides a data container to pass experimental data to the simulation routine for likelihood computation. when any of the properties are updated, the class automatically checks consistency of dimension and updates related properties and initialises them with NaNs.

Inheritance diagram for amidata:

Public Member Functions

- amidata (matlabtypesubstitute varargin)

  amidata creates an amidata container for experimental data with specified dimensions amidata.

Public Attributes

- matlabtypesubstitute nt = 0

  number of timepoints
- matlabtypesubstitute ny = 0

  number of observables
- matlabtypesubstitute nz = 0

  number of event observables
- matlabtypesubstitute ne = 0

  number of events
- matlabtypesubstitute nk = 0

  number of conditions/constants
- matlabtypesubstitute t = double.empty("")

  timepoints of observations
- matlabtypesubstitute Y = double.empty("")

  observations
- matlabtypesubstitute Sigma_Y = double.empty("")

  standard deviation of observations
- matlabtypesubstitute Z = double.empty("")

  event observations
- matlabtypesubstitute Sigma_Z = double.empty("")

  standard deviation of event observations
- matlabtypesubstitute condition = double.empty("")

  experimental condition
- matlabtypesubstitute conditionPreequilibration = double.empty("")

  experimental condition for preequilibration

10.25.1 Detailed Description

Definition at line 17 of file amidata.m.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 amidata()

```
amidata (
            matlabtypesubstitute varargin )
```

AMIDATA(amidata) creates a copy of the input container

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

fields

t [nt,1] Y [nt,ny] Sigma_Y [nt,ny] Z [ne,nz] Sigma_Z [ne,nz] condition [nk,1] conditionPreequilibration [nk,1] if some fields are missing the function will try to initialise them with NaNs with consistent dimensions

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

Parameters

varargin	

Definition at line 130 of file amidata.m.

10.25.3 Member Data Documentation

10.25.3.1 nt

```
nt = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 31 of file amidata.m.

10.25.3.2 ny

```
ny = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 39 of file amidata.m.

10.25.3.3 nz

```
nz = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 47 of file amidata.m.

10.25.3.4 ne

```
ne = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 55 of file amidata.m.

10.25.3.5 nk

```
nk = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 63 of file amidata.m.

10.25.3.6 t

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 71 of file amidata.m.

10.25.3.7 Y

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 79 of file amidata.m.

10.25.3.8 Sigma_Y

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 87 of file amidata.m.

10.25.3.9 Z

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 95 of file amidata.m.

10.25.3.10 Sigma_Z

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 103 of file amidata.m.

10.25.3.11 condition

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 111 of file amidata.m.

10.25.3.12 conditionPreequilibration

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

Default: double.empty("")

Note

This property has custom functionality when its value is changed.

Definition at line 119 of file amidata.m.

10.26 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 occurence
- ::symbolic z = sym.empty("")

  output function for the event
- matlabtypesubstitute hflag = logical.empty("")

  flag indicating that a heaviside function is present, this helps to speed up symbolic computations

10.26.1 Detailed Description

Definition at line 17 of file amievent.m.

10.26.2 Constructor & Destructor Documentation

10.26.2.1 amievent()

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

Parameters

trigger	trigger function, the event will be triggered on at all roots of this function
bolus	the bolus that will be added to all states on every occurence of the event
z	the event output that will be reported on every occurence of the event

Definition at line 75 of file amievent.m.

10.26.3 Member Function Documentation

10.26.3.1 setHflag()

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

Parameters

hflag	value for the hflag property

Return values

this	updated event definition object

Definition at line 18 of file setHflag.m.

Here is the caller graph for this function:

10.26.4 Member Data Documentation

10.26.4.1 trigger

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: sym.empty("")

Definition at line 27 of file amievent.m.

10.26.4.2 bolus

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: sym.empty("")

Definition at line 38 of file amievent.m.

10.26.4.3 z

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: sym.empty("")

Definition at line 49 of file amievent.m.

10.26.4.4 hflag

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: logical.empty("")

Definition at line 60 of file amievent.m.

10.27 amifun Class Reference

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

Public Member Functions

- amifun (matlabtypesubstitute funstr, matlabtypesubstitute model)

  amievent constructs an amifun object from the provided input.
- noret::substitute writeCcode_sensi (::amimodel model,::fileid fid)

  writeCcode_sensi is a wrapper for writeCcode which loops over parameters and reduces overhead by check nonzero values
- noret::substitute writeCcode (::amimodel model,::fileid fid)

  writeCcode is a wrapper for gccode which initialises data and reduces overhead by check nonzero values
- noret::substitute writeMcode (::amimodel model)

  writeMcode generates matlab evaluable code for specific model functions
- noret::substitute gccode (::amimodel model,::fileid fid)

  gccode transforms symbolic expressions into c code and writes the respective expression into a specified file
- mlhsInnerSubst< matlabtypesubstitute > getDeps (::amimodel model)

  getDeps populates the sensiflag for the requested function
- mlhsInnerSubst< matlabtypesubstitute > getArgs (::amimodel model)

  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 > getNVecs ()

  getfunargs populates the nvecs property with the names of the N_Vector elements which are required in the execution of the function (if applicable). the information is directly extracted from the argument string
- mlhsInnerSubst< matlabtypesubstitute > getCVar ()

  getCVar populates the cvar property
- mlhsInnerSubst< matlabtypesubstitute > getSensiFlag ()

  getSensiFlag populates the sensiflag property
- mlhsSubst< mlhsInnerSubst< matlabtypesubstitute >,mlhsInnerSubst<::amimodel > > getSyms (↩ ::amimodel model)

  getSyms computes the symbolic expression for the requested function

Public Attributes

- ::symbolic sym = sym("[ ]")

  symbolic definition struct
- ::symbolic sym_noopt = sym("[ ]")

  symbolic definition which was not optimized (no dependencies on w)
- ::symbolic strsym = sym("[ ]")

  short symbolic string which can be used for the reuse of precomputed values
- ::symbolic strsym_old = sym("[ ]")

  short symbolic string which can be used for the reuse of old values
- ::char funstr = char.empty("")

  name of the model
- ::char cvar = char.empty("")

  name of the c variable
- ::char argstr = char.empty("")

  argument string (solver specific)
- ::cell deps = cell.empty("")

  dependencies on other functions
- matlabtypesubstitute nvecs = cell.empty("")

  nvec dependencies
- matlabtypesubstitute sensiflag = logical.empty("")

  indicates whether the function is a sensitivity or derivative with respect to parameters

10.27.1 Detailed Description

Definition at line 17 of file amifun.m.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 amifun()

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

Parameters

funstr	name of the requested function
model	amimodel object which carries all symbolic definitions to construct the function

Definition at line 111 of file amifun.m.

10.27.3 Member Function Documentation

10.27.3.1 writeCcode_sensi()

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

Parameters

model	model defintion object
fid	file id in which the final expression is written

Return values

fid	void

Definition at line 18 of file writeCcode_sensi.m.

10.27.3.2 writeCcode()

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

Parameters

model	model defintion object
fid	file id in which the final expression is written

Return values

fid	void

Definition at line 18 of file writeCcode.m.

Here is the call graph for this function:

10.27.3.3 writeMcode()

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

Parameters

model	model defintion object

Return values

model	void

Definition at line 18 of file writeMcode.m.

10.27.3.4 gccode()

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

Parameters

model	model definition object
fid	file id in which the expression should be written

Return values

this	function definition object

Definition at line 18 of file gccode.m.

Here is the caller graph for this function:

10.27.3.5 getDeps()

```
mlhsInnerSubst<::amifun > getDeps (
            ::amimodel model )
```

Parameters

model	model definition object

Return values

this	updated function definition object

Definition at line 18 of file getDeps.m.

10.27.3.6 getArgs()

```
mlhsInnerSubst<::amifun > getArgs (
            ::amimodel model )
```

Parameters

model	model definition object

Return values

this	updated function definition object

Definition at line 18 of file getArgs.m.

10.27.3.7 getNVecs()

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

Return values

this	updated function definition object

Definition at line 18 of file getNVecs.m.

10.27.3.8 getCVar()

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

Return values

this	updated function definition object

Definition at line 18 of file getCVar.m.

10.27.3.9 getSensiFlag()

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

Return values

this	updated function definition object

Definition at line 18 of file getSensiFlag.m.

10.27.3.10 getSyms()

```
mlhsSubst< mlhsInnerSubst<::amifun >,mlhsInnerSubst<::amimodel > > getSyms (
            ::amimodel model )
```

Parameters

model	model definition object

Return values

this	updated function definition object
model	updated model definition object

Definition at line 18 of file getSyms.m.

10.27.4 Member Data Documentation

10.27.4.1 sym

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

Default: sym("[]")

Definition at line 27 of file amifun.m.

10.27.4.2 sym_noopt

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

Default: sym("[]")

Definition at line 35 of file amifun.m.

10.27.4.3 strsym

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

Default: sym("[]")

Definition at line 43 of file amifun.m.

10.27.4.4 strsym_old

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

Default: sym("[]")

Definition at line 51 of file amifun.m.

10.27.4.5 funstr

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

Default: char.empty("")

Definition at line 59 of file amifun.m.

10.27.4.6 cvar

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

Default: char.empty("")

Definition at line 67 of file amifun.m.

10.27.4.7 argstr

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

Default: char.empty("")

Definition at line 75 of file amifun.m.

10.27.4.8 deps

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

Default: cell.empty("")

Definition at line 83 of file amifun.m.

10.27.4.9 nvecs

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

Default: cell.empty("")

Definition at line 91 of file amifun.m.

10.27.4.10 sensiflag

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

Default: logical.empty("")

Definition at line 99 of file amifun.m.

10.28 amimodel Class Reference

AMIMODEL carries all model definitions including functions and events.

Inheritance diagram for amimodel:

Public Member Functions

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

  amimodel initializes the model object based on the provided symfun and modelname
- noret::substitute updateRHS (matlabtypesubstitute xdot)

  updateRHS updates the private fun property .fun.xdot.sym (right hand side of the differential equation)
- noret::substitute updateModelName (matlabtypesubstitute modelname)

  updateModelName updates the modelname
- noret::substitute updateWrapPath (matlabtypesubstitute wrap_path)

  updateModelName updates the modelname
- noret::substitute parseModel ()

  parseModel parses the model definition and computes all necessary symbolic expressions.
- noret::substitute generateC ()

  generateC generates the c files which will be used in the compilation.
- noret::substitute compileC ()

  compileC compiles the mex simulation file
- noret::substitute generateM (::amimodel amimodelo2)

  generateM generates the matlab wrapper for the compiled C files.
- noret::substitute getFun (::struct HTable,::string funstr)

  getFun generates symbolic expressions for the requested function.
- noret::substitute makeEvents ()

  makeEvents extracts discontiniuties 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 apporach 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 apporach later on.

Static Public Member Functions

- static noret::substitute compileAndLinkModel (matlabtypesubstitute modelname, matlabtypesubstitute modelSourceFolder, matlabtypesubstitute coptim, matlabtypesubstitute debug, matlabtypesubstitute funs, matlabtypesubstitute cfun)

  compileAndLinkModel compiles the mex simulation file. It does not check if the model files have changed since generating C++ code or whether all files are still present. Use only if you know what you are doing. The safer alternative is rerunning amiwrap().
- static noret::substitute generateMatlabWrapper (matlabtypesubstitute nx, matlabtypesubstitute ny, matlabtypesubstitute np, matlabtypesubstitute nk, matlabtypesubstitute nz, matlabtypesubstitute o2flag,::amimodel amimodelo2, matlabtypesubstitute wrapperFilename, matlabtypesubstitute modelname, matlabtypesubstitute pscale, matlabtypesubstitute forward, matlabtypesubstitute adjoint)

  generateMatlabWrapper generates the matlab wrapper for the compiled C files.

Public Attributes

- ::struct sym = struct.empty("")

  symbolic definition struct
- ::struct fun = struct.empty("")

  struct which stores information for which functions c code needs to be generated
- ::amievent event = amievent.empty("")

  struct which stores information for which functions c code needs to be generated
- ::string modelname = char.empty("")

  name of the model
- ::struct HTable = struct.empty("")

  struct that contains hash values for the symbolic model definitions
- ::bool debug = false

  flag indicating whether debugging symbols should be compiled
- ::bool adjoint = true

  flag indicating whether adjoint sensitivities should be enabled
- ::bool forward = true

  flag indicating whether forward sensitivities should be enabled
- ::double t0 = 0

  default initial time
- ::string wtype = char.empty("")

  type of wrapper (cvodes/idas)
- ::int nx = double.empty("")

  number of states
- ::int 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 mfuns = 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 occuring expressions

- ::int ndwdx = 0

  number of derivatives of derived variables w, dwdx

- ::int ndwdp = 0

  number of derivatives of derived variables w, dwdp

10.28.1 Detailed Description

Definition at line 17 of file amimodel.m.

10.28.2 Constructor & Destructor Documentation

10.28.2.1 amimodel()

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

Parameters

symfun	this is the string to the function which generates the modelstruct. You can also directly pass the struct here
modelname	name of the model

Definition at line 515 of file amimodel.m.

Here is the caller graph for this function:

10.28.3 Member Function Documentation

10.28.3.1 updateRHS()

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

Parameters

xdot	new right hand side of the differential equation

Return values

xdot	void

Definition at line 612 of file amimodel.m.

10.28.3.2 updateModelName()

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

Parameters

modelname	new modelname

Return values

modelname	void

Definition at line 627 of file amimodel.m.

10.28.3.3 updateWrapPath()

```
noret::substitute updateWrapPath (
            matlabtypesubstitute wrap_path )
```

Parameters

wrap_path	new wrap_path

Return values

wrap_path	void

Definition at line 640 of file amimodel.m.

10.28.3.4 parseModel()

```
noret::substitute parseModel ( )
```

Return values

void	

Definition at line 18 of file parseModel.m.

Here is the call graph for this function:

10.28.3.5 generateC()

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

Return values

void	

Definition at line 18 of file generateC.m.

10.28.3.6 compileC()

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

Return values

void	

Definition at line 18 of file compileC.m.

Here is the call graph for this function:

10.28.3.7 generateM()

```
noret::substitute generateM (
            ::amimodel amimodelo2 )
```

Parameters

amimodelo2	this struct must contain all necessary symbolic definitions for second order sensivities

Return values

amimodelo2	void

Definition at line 18 of file generateM.m.

Here is the call graph for this function:

10.28.3.8 getFun()

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

Parameters

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

Return values

funstr	void

Definition at line 18 of file getFun.m.

Here is the call graph for this function:

Here is the caller graph for this function:

10.28.3.9 makeEvents()

```
noret::substitute makeEvents ( )
```

Return values

void	

Definition at line 18 of file makeEvents.m.

Here is the call graph for this function:

10.28.3.10 makeSyms()

```
noret::substitute makeSyms ( )
```

Return values

void	

Definition at line 18 of file makeSyms.m.

10.28.3.11 checkDeps()

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

Parameters

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

Return values

cflag	boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable

Definition at line 18 of file checkDeps.m.

Here is the call graph for this function:

Here is the caller graph for this function:

10.28.3.12 loadOldHashes()

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

Return values

HTable	struct with hashes of symbolic definition from the previous compilation

Definition at line 18 of file loadOldHashes.m.

Here is the caller graph for this function:

10.28.3.13 augmento2()

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

Return values

this	augmented system which contains symbolic definition of the original system and its sensitivities

Definition at line 18 of file augmento2.m.

Here is the call graph for this function:

10.28.3.14 augmento2vec()

```
mlhsInnerSubst<::amimodel > augmento2vec ( )
```

Return values

modelo2vec	augmented system which contains symbolic definition of the original system and its sensitivities

Definition at line 18 of file augmento2vec.m.

Here is the call graph for this function:

10.28.3.15 compileAndLinkModel()

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

Parameters

modelname	name of the model as specified for amiwrap()
modelSourceFolder	path to model source directory
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

Return values

cfun	void

Definition at line 18 of file compileAndLinkModel.m.

Here is the caller graph for this function:

10.28.3.16 generateMatlabWrapper()

```
noret::substitute generateMatlabWrapper (
            matlabtypesubstitute nx,
            matlabtypesubstitute ny,
            matlabtypesubstitute np,
            matlabtypesubstitute nk,
            matlabtypesubstitute nz,
            matlabtypesubstitute o2flag,
            ::amimodel amimodelo2,
            matlabtypesubstitute wrapperFilename,
```

```
matlabtypesubstitute modelname,
matlabtypesubstitute pscale,
matlabtypesubstitute forward,
matlabtypesubstitute adjoint )  [static]
```

Parameters

nx	number of states
ny	number of observables
np	number of parameters
nk	number of fixed parameters
nz	number of events
o2flag	o2flag
amimodelo2	this struct must contain all necessary symbolic definitions for second order sensivities
wrapperFilename	output filename
modelname	name of the model
pscale	default parameter scaling
forward	has forward sensitivity equations
adjoint	has adjoint sensitivity equations

Return values

adjoint	void

Definition at line 18 of file generateMatlabWrapper.m.

Here is the caller graph for this function:

10.28.4 Member Data Documentation

10.28.4.1 sym

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 27 of file amimodel.m.

10.28.4.2 fun

```
fun = struct.empty("")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 38 of file amimodel.m.

10.28.4.3 event

```
event = amievent.empty("")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: amievent.empty("")

Definition at line 49 of file amimodel.m.

10.28.4.4 modelname

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: char.empty("")

Definition at line 61 of file amimodel.m.

10.28.4.5 HTable

```
HTable = struct.empty("")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 72 of file amimodel.m.

10.28.4.6 debug

```
debug = false
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: false

Definition at line 83 of file amimodel.m.

10.28.4.7 adjoint

```
adjoint = true
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: true

Definition at line 94 of file amimodel.m.

10.28.4.8 forward

```
forward = true
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: true

Definition at line 105 of file amimodel.m.

10.28.4.9 t0

```
t0 = 0
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: 0

Definition at line 116 of file amimodel.m.

10.28.4.10 wtype

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: char.empty("")

Definition at line 127 of file amimodel.m.

10.28.4.11 nx

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 138 of file amimodel.m.

10.28.4.12 nxtrue

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 149 of file amimodel.m.

10.28.4.13 ny

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 160 of file amimodel.m.

10.28.4.14 nytrue

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 171 of file amimodel.m.

10.28.4.15 np

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 182 of file amimodel.m.

10.28.4.16 nk

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 193 of file amimodel.m.

10.28.4.17 ng

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 204 of file amimodel.m.

10.28.4.18 nevent

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 215 of file amimodel.m.

10.28.4.19 nz

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 226 of file amimodel.m.

10.28.4.20 nztrue

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 237 of file amimodel.m.

10.28.4.21 id

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 248 of file amimodel.m.

10.28.4.22 ubw

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 259 of file amimodel.m.

10.28.4.23 lbw

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 270 of file amimodel.m.

10.28.4.24 nnz

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 281 of file amimodel.m.

10.28.4.25 sparseidx

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 292 of file amimodel.m.

10.28.4.26 rowvals

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 303 of file amimodel.m.

10.28.4.27 colptrs

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 314 of file amimodel.m.

10.28.4.28 sparseidxB

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 325 of file amimodel.m.

10.28.4.29 rowvalsB

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: double.empty("")

Definition at line 336 of file amimodel.m.

10.28.4.30 colptrsB

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: double.empty("")

Definition at line 347 of file amimodel.m.

10.28.4.31 funs

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: cell.empty("")

Definition at line 358 of file amimodel.m.

10.28.4.32 mfuns

```
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 369 of file amimodel.m.

10.28.4.33 coptim

```
coptim = "-O3"
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: "-O3"

Definition at line 380 of file amimodel.m.

10.28.4.34 param

```
param = "lin"
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: "lin"

Definition at line 391 of file amimodel.m.

10.28.4.35 wrap_path

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: char.empty("")

Definition at line 402 of file amimodel.m.

10.28.4.36 recompile

```
recompile = false
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: false

Definition at line 413 of file amimodel.m.

10.28.4.37 cfun

```
cfun = struct.empty("")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: struct.empty("")

Definition at line 424 of file amimodel.m.

10.28.4.38 o2flag

```
o2flag = 0
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: 0

Definition at line 436 of file amimodel.m.

10.28.4.39 z2event

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

Default: double.empty("")

Definition at line 455 of file amimodel.m.

10.28.4.40 splineflag

```
splineflag = false
```

Default: false

Definition at line 463 of file amimodel.m.

10.28.4.41 minflag

```
minflag = false
```

Default: false

Definition at line 471 of file amimodel.m.

10.28.4.42 maxflag

```
maxflag = false
```

Default: false

Definition at line 479 of file amimodel.m.

10.28.4.43 nw

```
nw = 0
```

Default: 0

Definition at line 487 of file amimodel.m.

10.28.4.44 ndwdx

```
ndwdx = 0
```

Default: 0

Definition at line 496 of file amimodel.m.

10.28.4.45 ndwdp

```
ndwdp = 0
```

Default: 0

Definition at line 504 of file amimodel.m.

10.29 amioption Class Reference

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

Inheritance diagram for amioption:

Public Member Functions

- amioption (matlabtypesubstitute varargin)

  amioptions Construct a new amioptions object OPTS = amioption() creates a set of options with each option set to its default value.

Public Attributes

- matlabtypesubstitute atol = 1e-16

  absolute integration tolerace
- matlabtypesubstitute rtol = 1e-8

  relative integration tolerace
- matlabtypesubstitute maxsteps = 1e4

  maximum number of integration steps
- matlabtypesubstitute quad_atol = 1e-12

  absolute integration tolerace
- matlabtypesubstitute quad_rtol = 1e-8

  relative integration tolerace
- matlabtypesubstitute maxstepsB = 0

  maximum number of integration steps
- matlabtypesubstitute sens_ind = double.empty("")

  index of parameters for which the sensitivities are computed
- matlabtypesubstitute tstart = 0

  starting time of the simulation
- matlabtypesubstitute lmm = 2

  linear multistep method.
- matlabtypesubstitute iter = 2

  iteration method for linear multistep.
- matlabtypesubstitute linsol = 9

  linear solver
- matlabtypesubstitute stldet = true

  stability detection flag
- matlabtypesubstitute interpType = 1

  interpolation type
- 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_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 ouputs 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`)

10.29.1 Detailed Description

Definition at line 17 of file amioption.m.

10.29.2 Constructor & Destructor Documentation

10.29.2.1 amioption()

```
amioption (
            matlabtypesubstitute varargin )
```

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

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

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

Parameters

varargin	input to construct amioption object, see function function description

Definition at line 242 of file amioption.m.

10.29.3 Member Data Documentation

10.29.3.1 atol

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

Default: 1e-16

Definition at line 28 of file amioption.m.

10.29.3.2 rtol

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

Default: 1e-8

Definition at line 36 of file amioption.m.

10.29.3.3 maxsteps

```
maxsteps = 1e4
```

Default: 1e4

Definition at line 44 of file amioption.m.

10.29.3.4 quad_atol

```
quad_atol = 1e-12
```

Default: 1e-12

Definition at line 52 of file amioption.m.

10.29.3.5 quad_rtol

```
quad_rtol = 1e-8
```

Default: 1e-8

Definition at line 60 of file amioption.m.

10.29.3.6 maxstepsB

```
maxstepsB = 0
```

Default: 0

Definition at line 68 of file amioption.m.

10.29.3.7 sens_ind

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

Default: double.empty("")

Definition at line 76 of file amioption.m.

10.29.3.8 tstart

```
tstart = 0
```

Default: 0

Definition at line 84 of file amioption.m.

10.29.3.9 lmm

```
lmm = 2
```

Default: 2

Definition at line 92 of file amioption.m.

10.29.3.10 iter

```
iter = 2
```

Default: 2

Definition at line 100 of file amioption.m.

10.29.3.11 linsol

```
linsol = 9
```

Default: 9

Definition at line 108 of file amioption.m.

10.29.3.12 stldet

```
stldet = true
```

Default: true

Definition at line 116 of file amioption.m.

10.29.3.13 interpType

```
interpType = 1
```

Default: 1

Definition at line 124 of file amioption.m.

10.29.3.14 ism

```
ism = 1
```

Default: 1

Definition at line 132 of file amioption.m.

10.29.3.15 sensi_meth

```
sensi_meth = 1
```

Default: 1

Note

This property has custom functionality when its value is changed.

Definition at line 140 of file amioption.m.

10.29.3.16 sensi

```
sensi = 0
```

Default: 0

Note

This property has custom functionality when its value is changed.

Definition at line 148 of file amioption.m.

10.29.3.17 nmaxevent

```
nmaxevent = 10
```

Default: 10

Definition at line 156 of file amioption.m.

10.29.3.18 ordering

```
ordering = 0
```

Default: 0

Definition at line 164 of file amioption.m.

10.29.3.19 ss

```
ss = 0
```

Default: 0

Definition at line 172 of file amioption.m.

10.29.3.20 x0

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

Default: double.empty("")

Definition at line 180 of file amioption.m.

10.29.3.21 sx0

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

Default: double.empty("")

Definition at line 188 of file amioption.m.

10.29.3.22 newton_maxsteps

```
newton_maxsteps = 40
```

Default: 40

Note

This property has custom functionality when its value is changed.

Definition at line 196 of file amioption.m.

10.29.3.23 newton_maxlinsteps

```
newton_maxlinsteps = 100
```

Default: 100

Note

This property has custom functionality when its value is changed.

Definition at line 204 of file amioption.m.

10.29.3.24 newton_preeq

```
newton_preeq = false
```

Default: false

Note

This property has custom functionality when its value is changed.

Definition at line 212 of file amioption.m.

10.29.3.25 z2event

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

Default: double.empty("")

Definition at line 220 of file amioption.m.

10.29.3.26 pscale

```
pscale = ""
```

Default: ""

Note

This property has custom functionality when its value is changed.

Definition at line 228 of file amioption.m.

10.30 amised Class Reference

AMISED is a container for SED-ML objects.

Inheritance diagram for amised:

Public Member Functions

- amised (matlabtypesubstitute sedname)

  amised reads in an SEDML document using the JAVA binding of of libSEDML

Public Attributes

- matlabtypesubstitute model = struct("'event',[ ],'sym',[ ]")

  amimodel from the specified model
- matlabtypesubstitute modelname = {""}

  cell array of model identifiers
- matlabtypesubstitute sedml = struct.empty("")

  stores the struct tree from the xml definition
- matlabtypesubstitute outputcount = "[ ]"

  count the number of outputs per model
- matlabtypesubstitute varidx = "[ ]"

  indexes for dataGenerators
- matlabtypesubstitute varsym = sym("[ ]")

  symbolic expressions for variables
- matlabtypesubstitute datasym = sym("[ ]")

  symbolic expressions for data

10.30.1 Detailed Description

Definition at line 17 of file amised.m.

10.30.2 Constructor & Destructor Documentation

10.30.2.1 amised()

```
amised (
            matlabtypesubstitute sedname )
```

Parameters

sedname	name/path of the SEDML document

Definition at line 112 of file amised.m.

Here is the call graph for this function:

10.30.3 Member Data Documentation

10.30.3.1 model

```
model = struct("'event',[],'sym',[]")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: struct("'event',[],'sym',[]")

Definition at line 27 of file amised.m.

10.30.3.2 modelname

```
modelname = {""}
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: {""}

Definition at line 38 of file amised.m.

10.30.3.3 sedml

```
sedml = struct.empty("")
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
`Matlab documentation of property attributes.`
Default: struct.empty("")

Definition at line 49 of file amised.m.

10.30.3.4 outputcount

```
outputcount = "[]"
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: "[]"

Definition at line 60 of file amised.m.

10.30.3.5 varidx

```
varidx = "[]"
```

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: "[]"

Definition at line 71 of file amised.m.

10.30.3.6 varsym

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: sym("[]")

Definition at line 82 of file amised.m.

10.30.3.7 datasym

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

Note

This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`
Matlab documentation of property attributes.
Default: sym("[]")

Definition at line 93 of file amised.m.

10.31 optsym Class Reference

OPTSYM is an auxiliary class to gain access to the private symbolic property `s` which is necessary to be able to call symobj::optimize on it.

Inheritance diagram for optsym:

Public Member Functions

- optsym (::sym symbol)

  optsym converts the symbolic object into a optsym object
- mlhsInnerSubst<::sym > getoptimized ()

  getoptimized calls symobj::optimize on the optsym object

10.31.1 Detailed Description

Definition at line 17 of file optsym.m.

10.31.2 Constructor & Destructor Documentation

10.31.2.1 optsym()

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

Parameters

symbol	symbolic object

Definition at line 32 of file optsym.m.

10.31.3 Member Function Documentation

10.31.3.1 getoptimized()

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

Return values

out	optimized symbolic object

Definition at line 42 of file optsym.m.

11 File Documentation

11.1 am_and.m File Reference

am_and is the amici implementation of the symbolic and function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_and (::sym a,::sym b)
  am_and is the amici implementation of the symbolic and function

11.1.1 Function Documentation

11.1.1.1 am_and()

```
mlhsInnerSubst< matlabtypesubstitute > am_and (
            ::sym a,
            ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_and.m.

Here is the call graph for this function:

Here is the caller graph for this function:

11.2 am_eq.m File Reference

am_eq is currently a placeholder that simply produces an error message

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_eq (matlabtypesubstitute varargin)
  am_eq is currently a placeholder that simply produces an error message

11.2.1 Function Documentation

11.2.1.1 am_eq()

```
mlhsInnerSubst< matlabtypesubstitute > am_eq (
            matlabtypesubstitute varargin )
```

Parameters

varargin	elements for chain of equalities

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_eq.m.

11.3 am_ge.m File Reference

am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_ge (::sym varargin)

  am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},...)

11.3.1 Function Documentation

11.3.1.1 am_ge()

```
mlhsInnerSubst< matlabtypesubstitute > am_ge (
            ::sym varargin )
```

Parameters

varargin	chain of input parameters

Return values

fun	a >= b logical value, negative for false, positive for true

Definition at line 17 of file am_ge.m.

Here is the call graph for this function:

11.4 am_gt.m File Reference

am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_gt (::sym varargin)

  am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

11.4.1 Function Documentation

11.4.1.1 am_gt()

```
mlhsInnerSubst< matlabtypesubstitute > am_gt (
            ::sym varargin )
```

Parameters

varargin	chain of input parameters

Return values

fun	a > b logical value, negative for false, positive for true

Definition at line 17 of file am_gt.m.

Here is the call graph for this function:

11.5 am_if.m File Reference

am_if is the amici implementation of the symbolic if function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_if (::sym condition,::sym truepart,::sym falsepart)

  am_if is the amici implementation of the symbolic if function

11.5.1 Function Documentation

11.5.1.1 am_if()

```
mlhsInnerSubst< matlabtypesubstitute > am_if (
            ::sym condition,
            ::sym truepart,
            ::sym falsepart )
```

Parameters

condition	logical value
truepart	value if condition is true
falsepart	value if condition is false

Return values

fun	if condition is true truepart, else falsepart

Definition at line 17 of file am_if.m.

Here is the caller graph for this function:

11.6 am_le.m File Reference

am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_le (::sym varargin)

  am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

11.6.1 Function Documentation

11.6.1.1 am_le()

```
mlhsInnerSubst< matlabtypesubstitute > am_le (
            ::sym varargin )
```

Parameters

varargin	chain of input parameters

Return values

fun	a <= b logical value, negative for false, positive for true

Definition at line 17 of file am_le.m.

Here is the call graph for this function:

11.7 am_lt.m File Reference

am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_lt (::sym varargin)

  am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},...)

11.7.1 Function Documentation

11.7.1.1 am_lt()

```
mlhsInnerSubst< matlabtypesubstitute > am_lt (
            ::sym varargin )
```

Parameters

varargin	chain of input parameters

Return values

fun	a < b logical value, negative for false, positive for true

Definition at line 17 of file am_lt.m.

Here is the call graph for this function:

11.8 am_max.m File Reference

am_max is the amici implementation of the symbolic max function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_max (::sym a,::sym b)

  am_max is the amici implementation of the symbolic max function

11.8.1 Function Documentation

11.8.1.1 am_max()

```
mlhsInnerSubst< matlabtypesubstitute > am_max (
            ::sym a,
            ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	maximum of a and b

Definition at line 17 of file am_max.m.

Here is the caller graph for this function:

11.9 am_min.m File Reference

am_min is the amici implementation of the symbolic min function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_min (::sym a,::sym b)

  am_min is the amici implementation of the symbolic min function

11.9.1 Function Documentation

11.9.1.1 am_min()

```
mlhsInnerSubst< matlabtypesubstitute > am_min (
            ::sym a,
            ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	minimum of a and b

Definition at line 17 of file am_min.m.

Here is the call graph for this function:

Here is the caller graph for this function:

11.10 am_or.m File Reference

am_or is the amici implementation of the symbolic or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_or (::sym a,::sym b)

  am_or is the amici implementation of the symbolic or function

11.10.1 Function Documentation

11.10.1.1 am_or()

```
mlhsInnerSubst< matlabtypesubstitute > am_or (
            ::sym a,
            ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_or.m.

Here is the call graph for this function:

Here is the caller graph for this function:

11.11 am_piecewise.m File Reference

am_piecewise is the amici implementation of the mathml piecewise function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_piecewise (matlabtypesubstitute piece, matlabtypesubstitute condition, matlabtypesubstitute default)

  am_piecewise is the amici implementation of the mathml piecewise function

11.11.1 Function Documentation

11.11.1.1 am_piecewise()

```
mlhsInnerSubst< matlabtypesubstitute > am_piecewise (
            matlabtypesubstitute piece,
            matlabtypesubstitute condition,
            matlabtypesubstitute default )
```

Parameters

piece	value if condition is true
condition	logical value
default	value if condition is false

Return values

fun	return value, piece if condition is true, default if not

Definition at line 17 of file am_piecewise.m.

Here is the call graph for this function:

11.12 am_stepfun.m File Reference

am_stepfun is the amici implementation of the step function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_stepfun (::sym t, matlabtypesubstitute tstart, matlabtypesubstitute vstart, matlabtypesubstitute tend, matlabtypesubstitute vend)

  am_stepfun is the amici implementation of the step function

11.12.1 Function Documentation

11.12.1.1 am_stepfun()

```
mlhsInnerSubst< matlabtypesubstitute > am_stepfun (
            ::sym t,
            matlabtypesubstitute tstart,
            matlabtypesubstitute vstart,
            matlabtypesubstitute tend,
            matlabtypesubstitute vend )
```

Parameters

t	input variable
tstart	input variable value at which the step starts
vstart	value during the step
tend	input variable value at which the step end
vend	value after the step

Return values

fun	0 before tstart, vstart between tstart and tend and vend after tend

Definition at line 17 of file am_stepfun.m.

11.13 am_xor.m File Reference

am_xor is the amici implementation of the symbolic exclusive or function

Functions

- mlhsInnerSubst< matlabtypesubstitute > am_xor (::sym a,::sym b)

  am_xor is the amici implementation of the symbolic exclusive or function

11.13.1 Function Documentation

11.13.1.1 am_xor()

```
mlhsInnerSubst< matlabtypesubstitute > am_xor (
            ::sym a,
            ::sym b )
```

Parameters

a	first input parameter
b	second input parameter

Return values

fun	logical value, negative for false, positive for true

Definition at line 17 of file am_xor.m.

Here is the call graph for this function:

11.14 amici.cpp File Reference

core routines for integration

```
#include "amici/amici.h"
#include "amici/backwardproblem.h"
#include "amici/forwardproblem.h"
#include "amici/misc.h"
#include <sundials/sundials_types.h>
#include <cvodes/cvodes.h>
#include <type_traits>
#include <cassert>
#include <cstdlib>
#include <cstring>
#include <cstdarg>
#include <memory>
#include <cmath>
```

Include dependency graph for amici.cpp:

Namespaces

- amici

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

Macros

- #define _USE_MATH_DEFINES
- #define M_PI 3.14159265358979323846

Functions

- std::unique_ptr< ReturnData > runAmiciSimulation (Solver &solver, const ExpData *edata, Model &model)
- void printErrMsgIdAndTxt (const char *identifier, const char *format,...)
- void printWarnMsgIdAndTxt (const char *identifier, const char *format,...)

11.14.1 Macro Definition Documentation

11.14.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

MS definition of PI and other constants

Definition at line 23 of file amici.cpp.

11.14.1.2 M_PI

```
#define M_PI 3.14159265358979323846
```

define PI if we still have no definition

Definition at line 27 of file amici.cpp.

11.15 amiwrap.m File Reference

AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

Functions

- noret::substitute amiwrap (matlabtypesubstitute varargin)

  AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

11.15.1 Function Documentation

11.15.1.1 amiwrap()

```
noret::substitute amiwrap (
            matlabtypesubstitute varargin )
```

Parameters

varargin	`amiwrap ( modelname, symfun, tdir, o2flag )` *Required Parameters for varargin:* • modelname specifies the name of the model which will be later used for the naming of the simulation file • symfun specifies a function which executes model definition see MATLAB Interface for details • tdir target directory where the simulation file should be placed **Default:** $AMICIDIR/models/modelname • o2flag boolean whether second order sensitivities should be enabled **Default:** false

Return values

o2flag	void

Definition at line 17 of file amiwrap.m.

11.16 cblas.cpp File Reference

BLAS routines required by AMICI.

```
#include "amici/defines.h"
#include <cblas.h>
```

Include dependency graph for cblas.cpp:

Namespaces

- amici

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

Functions

- void amici_dgemm (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, AMICI_BLAS_T↩ RANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double ∗A, const int lda, const double ∗B, const int ldb, const double beta, double ∗C, const int ldc)
- void amici_dgemv (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, const int M, const int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)

11.17 interface_matlab.cpp File Reference

core routines for mex interface

```
#include "amici/interface_matlab.h"
#include "amici/model.h"
#include "amici/exception.h"
#include "amici/edata.h"
#include "amici/returndata_matlab.h"
#include <assert.h>
#include <blas.h>
#include <cstring>
#include <memory>
```

Include dependency graph for interface_matlab.cpp:

Namespaces

- amici

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

Enumerations

- enum mexRhsArguments {
  RHS_TIMEPOINTS, **RHS_PARAMETERS**, **RHS_CONSTANTS**, **RHS_OPTIONS**, **RHS_PLIST**, **RHS_XSCALE_UNUSED**, **RHS_INITIALIZATION**, **RHS_DATA**, **RHS_NUMARGS_REQUIRED** = RHS_DATA, **RHS_NUMARGS** }

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

Functions

- int dbl2int (const double x)
- char amici_blasCBlasTransToBlasTrans (AMICI_BLAS_TRANSPOSE trans)
- void amici_dgemm (AMICI_BLAS_LAYOUT layout, AMICI_BLAS_TRANSPOSE TransA, AMICI_BLAS_T↩RANSPOSE 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 Model &model)
- void setSolverOptions (const mxArray ∗prhs[ ], int nrhs, Solver &solver)

  setSolverOptions solver options from the matlab call to a solver object
- void setModelData (const mxArray ∗prhs[ ], int nrhs, Model &model)

  setModelData sets data from the matlab call to the model object
- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

11.17.1 Detailed Description

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

11.17.2 Function Documentation

11.17.2.1 mexFunction()

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

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

Parameters

nlhs	number of output arguments of the matlab call
plhs	pointer to the array of output arguments
nrhs	number of input arguments of the matlab call
prhs	pointer to the array of input arguments

Returns

void

Definition at line 504 of file interface_matlab.cpp.

Here is the call graph for this function:

11.18 SBML2AMICI.m File Reference

SBML2AMICI generates AMICI model definition files from SBML.

Functions

- noret::substitute SBML2AMICI (matlabtypesubstitute filename, matlabtypesubstitute modelname)

  SBML2AMICI generates AMICI model definition files from SBML.

11.18.1 Function Documentation

11.18.1.1 SBML2AMICI()

```
noret::substitute SBML2AMICI (
            matlabtypesubstitute filename,
            matlabtypesubstitute modelname )
```

Parameters

filename	name of the SBML file (withouth extension)
modelname	name of the model, this will define the name of the output file (default: input filename)

Return values

modelname	void

Definition at line 17 of file SBML2AMICI.m.

Here is the caller graph for this function:

11.19 spline.cpp File Reference

definition of spline functions

Namespaces

- amici

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

Functions

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

  Evaluate the cubic spline function.
- double sinteg (int n, double u, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])

11.19.1 Detailed Description

Author

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

11.20 symbolic_functions.cpp File Reference

definition of symbolic functions

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

```
#include <alloca.h>
```

Include dependency graph for symbolic_functions.cpp:

Namespaces

- amici

  The AMICI Python module (in doxygen this will also contain documentation about the C++ library)

Functions

- int isNaN (double what)
- int isInf (double what)
- double getNaN ()
- double log (double x)
- double dirac (double x)
- double heaviside (double x)
- double sign (double x)
- double max (double a, double b, double c)
- double min (double a, double b, double c)
- double Dmax (int id, double a, double b, double c)
- double Dmin (int id, double a, double b, double c)
- double spline (double t, int num,...)
- double spline_pos (double t, int num,...)
- double Dspline (int id, double t, int num,...)
- double Dspline_pos (int id, double t, int num,...)
- double DDspline (int id1, int id2, double t, int num,...)
- double DDspline_pos (int id1, int id2, double t, int num,...)

11.20.1 Detailed Description

This file contains definitions of various symbolic functions which

Index

_USE_MATH_DEFINES
amici.cpp, 468
__init__
amici::sbml_import::SbmlImporter, 308
~AmiVector
amici::AmiVector, 74
~AmiVectorArray
amici::AmiVectorArray, 82
~ForwardProblem
amici::ForwardProblem, 102
~Model
amici::Model, 120

AMIAdjInit
amici::Solver, 379
AMIBand
amici::Solver, 382
AMIBandB
amici::Solver, 383
AMICI_BLAS_LAYOUT
amici, 18
AMICI_BLAS_TRANSPOSE
amici, 18
AMICI_o2mode
amici, 19
AMICI_parameter_scaling
amici, 19
AMICI_sensi_meth
amici, 19
AMICI_sensi_order
amici, 19
AMICalcICB
amici::Solver, 350
AMICalcIC
amici::Solver, 349
AMICreate
amici::Solver, 371
AMICreateB
amici::Solver, 380
AMIDense
amici::Solver, 381
AMIDenseB
amici::Solver, 382
AMIDiag
amici::Solver, 383
AMIDiagB
amici::Solver, 384
AMIFree
amici::Solver, 379
AMIGetAdjBmem
amici::Solver, 392
AMIGetDky
amici::Solver, 378
AMIGetLastOrder
amici::Solver, 391
AMIGetNumErrTestFails
amici::Solver, 390
AMIGetNumNonlinSolvConvFails
amici::Solver, 391
AMIGetNumRhsEvals
amici::Solver, 390
AMIGetNumSteps
amici::Solver, 389
AMIGetQuadB
amici::Solver, 354
AMIGetRootInfo
amici::Solver, 348
AMIGetSens
amici::Solver, 346
AMIGetB
amici::Solver, 353
AMIKLUSetOrdering
amici::Solver, 388
AMIKLUSetOrderingB
amici::Solver, 388
AMIKLUB
amici::Solver, 389
AMIKLU
amici::Solver, 387
AMIQuadReInitB
amici::Solver, 355
AMIQuadSStolerancesB
amici::Solver, 381
AMIReInit
amici::Solver, 348
AMIReInitB
amici::Solver, 353
AMISStolerances
amici::Solver, 372
AMISStolerancesB
amici::Solver, 380
AMISensReInit
amici::Solver, 349
AMISensSStolerances
amici::Solver, 372
AMISetErrHandlerFn
amici::Solver, 374
AMISetId
amici::Solver, 377
AMISetMaxNumSteps
amici::Solver, 375
AMISetMaxNumStepsB
amici::Solver, 375
AMISetQuadErrConB
amici::Solver, 373
AMISetSensErrCon
amici::Solver, 373
AMISetSensParams
amici::Solver, 378
AMISetStabLimDet

amici::Solver, 376
AMISetStabLimDetB
amici::Solver, 376
AMISetStopTime
amici::Solver, 352
AMISetSuppressAlg
amici::Solver, 377
AMISetUserData
amici::Solver, 374
AMISetUserDataB
amici::Solver, 375
AMISolve
amici::Solver, 350
AMISolveB
amici::Solver, 352
AMISolveF
amici::Solver, 351
AMISpbcg
amici::Solver, 385
AMISpbcgB
amici::Solver, 386
AMISpgmr
amici::Solver, 384
AMISpgmrB
amici::Solver, 385
AMISptfqmr
amici::Solver, 386
AMISptfqmrB
amici::Solver, 387
adjoint
amimodel, 432
am_and
am_and.m, 454
am_and.m, 454
am_and, 454
am_eq
am_eq.m, 455
am_eq.m, 455
am_eq, 455
am_ge
am_ge.m, 456
am_ge.m, 456
am_ge, 456
am_gt
am_gt.m, 457
am_gt.m, 457
am_gt, 457
am_if
am_if.m, 458
am_if.m, 458
am_if, 458
am_le
am_le.m, 459
am_le.m, 459
am_le, 459
am_lt
am_lt.m, 460
am_lt.m, 460
am_lt, 460
am_max
am_max.m, 461
am_max.m, 461
am_max, 461
am_min
am_min.m, 462
am_min.m, 462
am_min, 462
am_or
am_or.m, 463
am_or.m, 463
am_or, 463
am_piecewise
am_piecewise.m, 464
am_piecewise.m, 464
am_piecewise, 464
am_stepfun
am_stepfun.m, 465
am_stepfun.m, 465
am_stepfun, 465
am_xor
am_xor.m, 466
am_xor.m, 466
am_xor, 466
ami_mem
amici::Solver, 397
AmiException, 70
amici::AmiException, 71
AmiVector, 73
amici::AmiException, 73, 74
AmiVectorArray, 80
amici::AmiVectorArray, 81
amici, 14
AMICI_BLAS_LAYOUT, 18
AMICI_BLAS_TRANSPOSE, 18
AMICI_o2mode, 19
AMICI_parameter_scaling, 19
AMICI_sensi_meth, 19
AMICI_sensi_order, 19
amici_blasCBlasTransToBlasTrans, 65
amici_dgemm, 24
amici_dgemv, 22
amici_path, 66
checkFieldNames, 41
checkFinite, 29
DDspline, 61
DDspline_pos, 61
dbl2int, 65
deserializeFromChar, 43
deserializeFromString, 44
dirac, 49
Dmax, 53
Dmin, 50
Dspline, 59
Dspline_pos, 60
errMsgIdAndTxt, 66
expDataFromMatlabCall, 28

getFieldAsNumPyArray, 64
getNaN, 56
getReturnDataMatlabFromAmiciCall, 30
heaviside, 49
initAndAttachArray, 40
initMatlabDiagnosisFields, 32
initMatlabReturnFields, 31
InternalSensitivityMethod, 20
InterpolationType, 20
isInf, 55
isNaN, 54
LinearMultistepMethod, 20
LinearSolver, 19
log, 48
max, 52
min, 50
msgIdAndTxtFp, 18
NonlinearSolverIteration, 20
operator==, 30, 44
pi, 66
printErrMsgIdAndTxt, 20
printWarnMsgIdAndTxt, 21
rdataToNumPyArrays, 63
realtype, 18
reorder, 41
runAmiciSimulation, 21, 62
serializeToChar, 42
serializeToStdVec, 43
serializeToString, 43
setModelData, 26
setSolverOptions, 27
setupReturnData, 27
seval, 46
sign, 57
sinteg, 47
spline, 45, 57
spline_pos, 58
StateOrdering, 20
warnMsgIdAndTxt, 66
writeMatlabField0, 34
writeMatlabField1, 35
writeMatlabField2, 36
writeMatlabField3, 37
writeMatlabField4, 38
amici.cpp, 467
_USE_MATH_DEFINES, 468
M_PI, 468
amici.sbml_import, 66
amici::AmiException
AmiException, 71
getBacktrace, 72
storeBacktrace, 72
what, 71
amici::AmiVector
∼AmiVector, 74
AmiVector, 73, 74
at, 80
data, 75, 76
getLength, 78
getNVector, 76
getVector, 77
minus, 78
operator=, 75
operator[], 79
reset, 78
set, 79
amici::AmiVectorArray
∼AmiVectorArray, 82
AmiVectorArray, 81
at, 83
data, 82, 83
getLength, 86
getNVector, 84
getNVectorArray, 84
operator[], 85
reset, 86
amici::BackwardProblem
BackwardProblem, 87
getdJydx, 92
getdxBptr, 91
gett, 89
getwhich, 90
getwhichptr, 90
getxBptr, 90
getxQBptr, 91
workBackwardProblem, 88
amici::CvodeException
CvodeException, 93
amici::ExpData
ExpData, 94–96
fixedParameters, 100
fixedParametersPreequilibration, 100
my, 98
mz, 99
nmaxevent, 100
nt, 99
nytrue, 99
nztrue, 99
setObservedData, 96
setObservedDataStdDev, 97
setObservedEvents, 97
setObservedEventsStdDev, 98
sigmay, 98
sigmaz, 99
amici::ForwardProblem
∼ForwardProblem, 102
edata, 109
ForwardProblem, 101
getDJydx, 106
getDJzdx, 106
getDiscontinuities, 105
getNumberOfRoots, 105
getRHSAtDiscontinuities, 104
getRHSBeforeDiscontinuities, 105
getRootCounter, 106
getRootIndexes, 105

getStateDerivativePointer, 107
getStateDerivativeSensitivityPointer, 108
getStatePointer, 107
getStateSensitivity, 104
getStateSensitivityPointer, 108
getStatesAtDiscontinuities, 104
getTime, 103
model, 109
rdata, 109
solver, 109
workForwardProblem, 102
amici::IDAException
IDAException, 110
amici::IntegrationFailure
error_code, 112
IntegrationFailure, 112
time, 112
amici::Model
~Model, 120
boost::serialization::serialize, 201
checkFinite, 175
clone, 120
dJrzdsigma, 209
dJrzdz, 209
dJydp, 206
dJydsigma, 208
dJydy, 208
dJzdp, 206
dJzdsigma, 209
dJzdz, 208
deltaqB, 207
deltasx, 207
deltax, 206
deltaxB, 207
drzdp, 210
drzdx, 210
dsigmaydp, 205
dsigmazdp, 206
dwdp, 211
dwdx, 211
dxdotdp, 207
dydp, 210
dydx, 210
dzdp, 209
dzdx, 209
fJDiag, 123
fJSparse, 123
fJrz, 143, 189
fJv, 125
fJy, 142, 188
fJz, 142, 188
fdJrzdsigma, 147, 192
fdJrzdz, 147, 191
fdJydp, 150
fdJydsigma, 144, 190
fdJydx, 151
fdJydy, 144, 189
fdJzdp, 152
fdJzdsigma, 146, 191
fdJzdx, 153
fdJzdz, 145, 190
fdeltaqB, 139, 185
fdeltasx, 137, 184
fdeltax, 136, 184
fdeltaxB, 138, 185
fdrzdp, 135, 183
fdrzdx, 136, 183
fdsigma_ydp, 140, 187
fdsigma_zdp, 141, 187
fdwdp, 171, 193
fdwdx, 172, 193
fdx0, 126
fdxdotdp, 124
fdydp, 129, 178
fdydx, 130, 179
fdzdp, 134, 181
fdzdx, 135, 182
fixedParameters, 212
fJ, 122
froot, 121
frz, 132, 180
fsJy, 149
fsJz, 151
fsdx0, 128
fsigma_y, 140, 186
fsigma_z, 140, 187
fsrz, 133, 181
fstau, 128, 177
fsx0, 127, 177
fsy, 148
fsz, 131, 180
fsz_tf, 148
fw, 170, 192
fx0, 126, 176
fxdot, 122
fy, 129, 178
fz, 131, 179
getFixedParameters, 162
getInitialStateSensitivities, 167
getInitialStates, 166
getParameterList, 165
getParameterScale, 160
getParameters, 161
getSolver, 121
getTimepoints, 163
getUnscaledParameters, 162
getmy, 194
getmz, 195
getrz, 199
getsrz, 200
getsx, 198
getsz, 200
gett, 175
getx, 197
gety, 196
getz, 198

h, 211
idlist, 205
initHeaviside, 155
initialize, 154
initializeStates, 154
J, 207
k, 158
lbw, 204
M, 211
Model, 118–120
my, 208
mz, 208
nMaxEvent, 159
ndwdp, 204
ndwdx, 203
ne, 203
nJ, 204
nk, 158
nmaxevent, 213
nnz, 204
np, 157
nplist, 156
nt, 159
nw, 203
nx, 202
nxtrue, 202
ny, 202
nytrue, 202
nz, 203
nztrue, 203
o2mode, 205
operator=, 120
operator==, 201
originalParameters, 212
plist, 169
plist_, 212
pscale, 213
setFixedParameters, 162
setInitialStateSensitivities, 167
setInitialStates, 166
setNMaxEvent, 159
setParameterList, 166
setParameterScale, 160
setParameters, 161
setT0, 168
setTimepoints, 163
sigmay, 205
sigmaz, 206
stau, 211
sx0data, 213
t, 164
t0, 167
ts, 213
tstart, 213
ubw, 204
unscaleParameters, 169
unscaledParameters, 212
updateHeaviside, 174
updateHeavisideB, 175
w, 210
x0data, 212
z2event, 205
amici::Model_DAE
fJDiag, 222, 236
fJSparse, 219, 220, 235
fJSparseB, 221, 235
fJB, 218, 234
fJv, 222, 223, 237
fJvB, 224, 237
fdxdotdp, 230, 231, 240
fJ, 217, 218, 233
fM, 232, 242
fqBdot, 229, 240
froot, 225, 226, 238
fsxdot, 231, 241
fxBdot, 228, 239
fxdot, 227, 228, 239
getSolver, 233
Model_DAE, 215
amici::Model_ODE
fJDiag, 251, 252, 265
fJSparse, 248, 249, 264
fJSparseB, 250, 265
fJB, 248, 264
fJv, 253, 254, 266
fJvB, 255, 266
fdxdotdp, 260, 261, 269
fJ, 246, 247, 263
fqBdot, 259, 268
froot, 255, 256, 267
fsxdot, 262, 270
fxBdot, 259, 268
fxdot, 257, 258, 267
getSolver, 263
Model_ODE, 244, 245
amici::NewtonFailure
NewtonFailure, 271
amici::NewtonSolver
atol, 277
dx, 278
getSensis, 275
getSolver, 273
getStep, 274
maxlinsteps, 277
maxsteps, 277
model, 277
NewtonSolver, 273
prepareLinearSystem, 275
rdata, 278
rtol, 277
solveLinearSystem, 276
t, 277
x, 278
xdot, 278
amici::NewtonSolverDense
NewtonSolverDense, 279

prepareLinearSystem, 280
solveLinearSystem, 280
amici::NewtonSolverIterative
linsolveSPBCG, 284
NewtonSolverIterative, 282
prepareLinearSystem, 283
solveLinearSystem, 283
amici::NewtonSolverSparse
NewtonSolverSparse, 286
prepareLinearSystem, 287
solveLinearSystem, 287
amici::ReturnData
applyChainRuleFactorToSimulationResults, 292
boost::serialization::serialize, 293
chi2, 300
invalidate, 290
invalidateLLH, 292
J, 294
llh, 300
ne, 302
newton_maxsteps, 303
newton_numlinsteps, 299
newton_numsteps, 299
newton_status, 299
newton_time, 299
nJ, 303
nk, 301
nmaxevent, 303
np, 301
nplist, 303
nt, 303
numerrtestfails, 298
numerrtestfailsB, 298
numnonlinsolvconvfails, 298
numnonlinsolvconvfailsB, 298
numrhsevals, 297
numrhsevalsB, 297
numsteps, 297
numstepsB, 297
nx, 301
nxtrue, 301
ny, 302
nytrue, 302
nz, 302
nztrue, 302
o2mode, 304
order, 298
pscale, 304
res, 296
ReturnData, 290
rz, 295
s2llh, 300
s2rz, 295
sensi, 304
sensi_meth, 304
sigmay, 296
sigmaz, 294
sllh, 300
sres, 297
srz, 295
ssigmay, 296
ssigmaz, 294
status, 301
sx, 295
sx0, 300
sy, 296
sz, 294
ts, 293
x, 295
x0, 299
xdot, 294
y, 296
z, 294
amici::SetupFailure
SetupFailure, 337
amici::Solver
AMIAdjInit, 379
AMIBand, 382
AMIBandB, 383
AMICalcICB, 350
AMICalcIC, 349
AMICreate, 371
AMICreateB, 380
AMIDense, 381
AMIDenseB, 382
AMIDiag, 383
AMIDiagB, 384
AMIFree, 379
AMIGetAdjBmem, 392
AMIGetDky, 378
AMIGetLastOrder, 391
AMIGetNumErrTestFails, 390
AMIGetNumNonlinSolvConvFails, 391
AMIGetNumRhsEvals, 390
AMIGetNumSteps, 389
AMIGetQuadB, 354
AMIGetRootInfo, 348
AMIGetSens, 346
AMIGetB, 353
AMIKLUSetOrdering, 388
AMIKLUSetOrderingB, 388
AMIKLUB, 389
AMIKLU, 387
AMIQuadReInitB, 355
AMIQuadSStolerancesB, 381
AMIReInit, 348
AMIReInitB, 353
AMISStolerances, 372
AMISStolerancesB, 380
AMISensReInit, 349
AMISensSStolerances, 372
AMISetErrHandlerFn, 374
AMISetId, 377
AMISetMaxNumSteps, 375
AMISetMaxNumStepsB, 375
AMISetQuadErrConB, 373

AMISetSensErrCon, 373
AMISetSensParams, 378
AMISetStabLimDet, 376
AMISetStabLimDetB, 376
AMISetStopTime, 352
AMISetSuppressAlg, 377
AMISetUserData, 374
AMISetUserDataB, 375
AMISolve, 350
AMISolveB, 352
AMISolveF, 351
AMISpbcg, 385
AMISpbcgB, 386
AMISpgmr, 384
AMISpgmrB, 385
AMISptfqmr, 386
AMISptfqmrB, 387
ami_mem, 397
binit, 366
boost::serialization::serialize, 397
clone, 342
getAbsoluteTolerance, 359
getAbsoluteToleranceQuadratures, 360
getDiagnosis, 346
getDiagnosisB, 347
getInternalSensitivityMethod, 365
getInterpolationType, 362
getLinearMultistepMethod, 361
getLinearSolver, 364
getMaxSteps, 360
getMaxStepsBackwardProblem, 361
getNewtonMaxLinearSteps, 357
getNewtonMaxSteps, 356
getNewtonPreequilibration, 357
getNonlinearSolverIteration, 362
getRelativeTolerance, 358
getRelativeToleranceQuadratures, 359
getSensitivityMethod, 356
getSensitivityOrder, 358
getStabilityLimitFlag, 364
getStateOrdering, 363
init, 365
initializeLinearSolver, 393
initializeLinearSolverB, 395
operator==, 397
qbinit, 366
rootInit, 367
sensInit1, 367
setAbsoluteTolerance, 359
setAbsoluteToleranceQuadratures, 360
setBandJacFn, 368
setBandJacFnB, 370
setDenseJacFn, 368
setDenseJacFnB, 369
setInternalSensitivityMethod, 365
setInterpolationType, 363
setJacTimesVecFn, 369
setJacTimesVecFnB, 370
setLinearMultistepMethod, 362
setLinearSolver, 364
setMaxSteps, 361
setMaxStepsBackwardProblem, 361
setNewtonMaxLinearSteps, 358
setNewtonMaxSteps, 356
setNewtonPreequilibration, 357
setNonlinearSolverIteration, 362
setRelativeTolerance, 359
setRelativeToleranceQuadratures, 360
setSensitivityMethod, 356
setSensitivityOrder, 358
setSparseJacFn, 368
setSparseJacFnB, 369
setStabilityLimitFlag, 364
setStateOrdering, 363
setupAMIB, 344
setupAMI, 342
Solver, 341
solverWasCalled, 398
turnOffRootFinding, 355
wrapErrHandlerFn, 371
amici::SteadystateProblem
applyNewtonsMethod, 400
getNewtonOutput, 401
getNewtonSimulation, 401
SteadystateProblem, 398
workSteadyStateProblem, 399
amici::sbml_import
applyTemplate, 67
assignmentRules2observables, 69
getRuleVars, 69
getSymbolicDiagonal, 68
getSymbols, 68
amici::sbml_import::SbmlImporter
__init__, 308
checkSupport, 313
cleanReservedSymbols, 318
compileCCode, 326
computeModelEquations, 320
computeModelEquationsAdjointSensitivites, 324
computeModelEquationsForwardSensitivites, 323
computeModelEquationsLinearSolver, 321
computeModelEquationsObjectiveFunction, 321
computeModelEquationsSensitivitesCore, 323
generateCCode, 325
getFunctionBody, 328
getSparseSymLines, 334
getSparseSymbols, 319
getSymLines, 333
loadSBMLFile, 309
prepareModelFolder, 325
printWithException, 335
processCompartments, 315
processParameters, 314
processReactions, 315
processRules, 315
processSBML, 312

processSpecies, 314
processTime, 316
processVolumeConversion, 316
replaceInAllExpressions, 317
replaceSpecialConstants, 318
sbml2amici, 309
setName, 310
setPaths, 311
writeCMakeFile, 331
writeFunctionFile, 327
writeIndexFiles, 327
writeModelHeader, 330
writeModuleSetup, 332
writeSwigFiles, 332
writeWrapfunctionsCPP, 329
writeWrapfunctionsHeader, 330
amici_blasCBlasTransToBlasTrans
amici, 65
amici_dgemm
amici, 24
amici_dgemv
amici, 22
amici_path
amici, 66
amidata, 402
amidata, 404
condition, 406
conditionPreequilibration, 407
ne, 405
nk, 405
nt, 404
ny, 404
nz, 404
Sigma_Y, 406
Sigma_Z, 406
t, 405
Y, 405
Z, 406
amievent, 407
amievent, 408
bolus, 409
hflag, 409
setHflag, 408
trigger, 409
z, 409
amifun, 410
amifun, 411
argstr, 416
cvar, 416
deps, 417
funstr, 416
gccode, 413
getArgs, 414
getCVar, 414
getDeps, 413
getNVecs, 414
getSensiFlag, 415
getSyms, 415
nvecs, 417
sensiflag, 417
strsym, 416
strsym_old, 416
sym, 415
sym_noopt, 415
writeCcode, 412
writeCcode_sensi, 411
writeMcode, 412
amimodel, 417
adjoint, 432
amimodel, 421
augmento2, 428
augmento2vec, 428
cfun, 439
checkDeps, 426
colptrs, 437
colptrsB, 437
compileAndLinkModel, 429
compileC, 423
coptim, 438
debug, 431
event, 431
forward, 432
fun, 430
funs, 438
generateMatlabWrapper, 429
generateC, 423
generateM, 423
getFun, 424
HTable, 431
id, 435
lbw, 436
loadOldHashes, 427
makeEvents, 425
makeSyms, 426
maxflag, 440
mfuns, 438
minflag, 440
modelname, 431
ndwdp, 441
ndwdx, 441
nevent, 434
ng, 434
nk, 434
nnz, 436
np, 434
nw, 440
nx, 433
nxtrue, 433
ny, 433
nytrue, 433
nz, 435
nztrue, 435
o2flag, 439
param, 438
parseModel, 422
recompile, 439

rowvals, 436
rowvalsB, 437
sparseidx, 436
sparseidxB, 437
splineflag, 440
sym, 430
t0, 432
ubw, 435
updateModelName, 422
updateRHS, 421
updateWrapPath, 422
wrap_path, 439
wtype, 432
z2event, 440
amioption, 441
amioption, 443
atol, 443
interpType, 446
ism, 446
iter, 445
linsol, 445
lmm, 445
maxsteps, 444
maxstepsB, 444
newton_maxlinsteps, 448
newton_maxsteps, 448
newton_preeq, 448
nmaxevent, 447
ordering, 447
pscale, 449
quad_atol, 444
quad_rtol, 444
rtol, 443
sens_ind, 444
sensi, 446
sensi_meth, 446
ss, 447
stldet, 445
sx0, 447
tstart, 445
x0, 447
z2event, 448
amised, 449
amised, 450
datasym, 452
model, 451
modelname, 451
outputcount, 451
sedml, 451
varidx, 452
varsym, 452
amiwrap
amiwrap.m, 468
amiwrap.m, 468
amiwrap, 468
applyChainRuleFactorToSimulationResults
amici::ReturnData, 292
applyNewtonsMethod
amici::SteadystateProblem, 400
applyTemplate
amici::sbml_import, 67
argstr
amifun, 416
assignmentRules2observables
amici::sbml_import, 69
at
amici::AmiVector, 80
amici::AmiVectorArray, 83
atol
amici::NewtonSolver, 277
amioption, 443
augmento2
amimodel, 428
augmento2vec
amimodel, 428

BackwardProblem, 87
amici::BackwardProblem, 87
binit
amici::Solver, 366
bolus
amievent, 409
boost::serialization::serialize
amici::Model, 201
amici::ReturnData, 293
amici::Solver, 397

cblas.cpp, 469
cfun
amimodel, 439
checkDeps
amimodel, 426
checkFieldNames
amici, 41
checkFinite
amici, 29
amici::Model, 175
checkSupport
amici::sbml_import::SbmlImporter, 313
chi2
amici::ReturnData, 300
cleanReservedSymbols
amici::sbml_import::SbmlImporter, 318
clone
amici::Model, 120
amici::Solver, 342
colptrs
amimodel, 437
colptrsB
amimodel, 437
compileAndLinkModel
amimodel, 429
compileCCode
amici::sbml_import::SbmlImporter, 326
compileC
amimodel, 423
computeModelEquations

amici::sbml_import::SbmlImporter, 320
computeModelEquationsAdjointSensitivites
amici::sbml_import::SbmlImporter, 324
computeModelEquationsForwardSensitivites
amici::sbml_import::SbmlImporter, 323
computeModelEquationsLinearSolver
amici::sbml_import::SbmlImporter, 321
computeModelEquationsObjectiveFunction
amici::sbml_import::SbmlImporter, 321
computeModelEquationsSensitivitesCore
amici::sbml_import::SbmlImporter, 323
condition
amidata, 406
conditionPreequilibration
amidata, 407
coptim
amimodel, 438
cvar
amifun, 416
CvodeException, 93
amici::CvodeException, 93

DDspline
amici, 61
DDspline_pos
amici, 61
dJrzdsigma
amici::Model, 209
dJrzdz
amici::Model, 209
dJydp
amici::Model, 206
dJydsigma
amici::Model, 208
dJydy
amici::Model, 208
dJzdp
amici::Model, 206
dJzdsigma
amici::Model, 209
dJzdz
amici::Model, 208
data
amici::AmiVector, 75, 76
amici::AmiVectorArray, 82, 83
datasym
amised, 452
dbl2int
amici, 65
debug
amimodel, 431
deltaqB
amici::Model, 207
deltasx
amici::Model, 207
deltax
amici::Model, 206
deltaxB
amici::Model, 207
deps
amifun, 417
deserializeFromChar
amici, 43
deserializeFromString
amici, 44
dirac
amici, 49
Dmax
amici, 53
Dmin
amici, 50
drzdp
amici::Model, 210
drzdx
amici::Model, 210
dsigmaydp
amici::Model, 205
dsigmazdp
amici::Model, 206
Dspline
amici, 59
Dspline_pos
amici, 60
dwdp
amici::Model, 211
dwdx
amici::Model, 211
dx
amici::NewtonSolver, 278
dxdotdp
amici::Model, 207
dydp
amici::Model, 210
dydx
amici::Model, 210
dzdp
amici::Model, 209
dzdx
amici::Model, 209

edata
amici::ForwardProblem, 109
errMsgIdAndTxt
amici, 66
error_code
amici::IntegrationFailure, 112
event
amimodel, 431
ExpData, 94
amici::ExpData, 94–96
expDataFromMatlabCall
amici, 28

fJDiag
amici::Model, 123
amici::Model_DAE, 222, 236
amici::Model_ODE, 251, 252, 265
fJSparse

amici::Model, 123
amici::Model_DAE, 219, 220, 235
amici::Model_ODE, 248, 249, 264
fJSparseB
amici::Model_DAE, 221, 235
amici::Model_ODE, 250, 265
fJB
amici::Model_DAE, 218, 234
amici::Model_ODE, 248, 264
fJrz
amici::Model, 143, 189
fJv
amici::Model, 125
amici::Model_DAE, 222, 223, 237
amici::Model_ODE, 253, 254, 266
fJvB
amici::Model_DAE, 224, 237
amici::Model_ODE, 255, 266
fJy
amici::Model, 142, 188
fJz
amici::Model, 142, 188
fdJrzdsigma
amici::Model, 147, 192
fdJrzdz
amici::Model, 147, 191
fdJydp
amici::Model, 150
fdJydsigma
amici::Model, 144, 190
fdJydx
amici::Model, 151
fdJydy
amici::Model, 144, 189
fdJzdp
amici::Model, 152
fdJzdsigma
amici::Model, 146, 191
fdJzdx
amici::Model, 153
fdJzdz
amici::Model, 145, 190
fdeltaqB
amici::Model, 139, 185
fdeltasx
amici::Model, 137, 184
fdeltax
amici::Model, 136, 184
fdeltaxB
amici::Model, 138, 185
fdrzdp
amici::Model, 135, 183
fdrzdx
amici::Model, 136, 183
fdsigma_ydp
amici::Model, 140, 187
fdsigma_zdp
amici::Model, 141, 187
fdwdp
amici::Model, 171, 193
fdwdx
amici::Model, 172, 193
fdx0
amici::Model, 126
fdxdotdp
amici::Model, 124
amici::Model_DAE, 230, 231, 240
amici::Model_ODE, 260, 261, 269
fdydp
amici::Model, 129, 178
fdydx
amici::Model, 130, 179
fdzdp
amici::Model, 134, 181
fdzdx
amici::Model, 135, 182
fixedParameters
amici::ExpData, 100
amici::Model, 212
fixedParametersPreequilibration
amici::ExpData, 100
fJ
amici::Model, 122
amici::Model_DAE, 217, 218, 233
amici::Model_ODE, 246, 247, 263
fM
amici::Model_DAE, 232, 242
forward
amimodel, 432
ForwardProblem, 100
amici::ForwardProblem, 101
fqBdot
amici::Model_DAE, 229, 240
amici::Model_ODE, 259, 268
froot
amici::Model, 121
amici::Model_DAE, 225, 226, 238
amici::Model_ODE, 255, 256, 267
frz
amici::Model, 132, 180
fsJy
amici::Model, 149
fsJz
amici::Model, 151
fsdx0
amici::Model, 128
fsigma_y
amici::Model, 140, 186
fsigma_z
amici::Model, 140, 187
fsrz
amici::Model, 133, 181
fstau
amici::Model, 128, 177
fsx0
amici::Model, 127, 177

fsxdot
amici::Model_DAE, 231, 241
amici::Model_ODE, 262, 270
fsy
amici::Model, 148
fsz
amici::Model, 131, 180
fsz_tf
amici::Model, 148
fun
amimodel, 430
funs
amimodel, 438
funstr
amifun, 416
fw
amici::Model, 170, 192
fx0
amici::Model, 126, 176
fxBdot
amici::Model_DAE, 228, 239
amici::Model_ODE, 259, 268
fxdot
amici::Model, 122
amici::Model_DAE, 227, 228, 239
amici::Model_ODE, 257, 258, 267
fy
amici::Model, 129, 178
fz
amici::Model, 131, 179

gccode
amifun, 413
generateCCode
amici::sbml_import::SbmlImporter, 325
generateMatlabWrapper
amimodel, 429
generateC
amimodel, 423
generateM
amimodel, 423
getAbsoluteTolerance
amici::Solver, 359
getAbsoluteToleranceQuadratures
amici::Solver, 360
getArgs
amifun, 414
getBacktrace
amici::AmiException, 72
getCVar
amifun, 414
getDJydx
amici::ForwardProblem, 106
getDJzdx
amici::ForwardProblem, 106
getDeps
amifun, 413
getDiagnosis
amici::Solver, 346
getDiagnosisB
amici::Solver, 347
getDiscontinuities
amici::ForwardProblem, 105
getFieldAsNumPyArray
amici, 64
getFixedParameters
amici::Model, 162
getFun
amimodel, 424
getFunctionBody
amici::sbml_import::SbmlImporter, 328
getInitialStateSensitivities
amici::Model, 167
getInitialStates
amici::Model, 166
getInternalSensitivityMethod
amici::Solver, 365
getInterpolationType
amici::Solver, 362
getLength
amici::AmiVector, 78
amici::AmiVectorArray, 86
getLinearMultistepMethod
amici::Solver, 361
getLinearSolver
amici::Solver, 364
getMaxSteps
amici::Solver, 360
getMaxStepsBackwardProblem
amici::Solver, 361
getNVecs
amifun, 414
getNVector
amici::AmiVector, 76
amici::AmiVectorArray, 84
getNVectorArray
amici::AmiVectorArray, 84
getNaN
amici, 56
getNewtonMaxLinearSteps
amici::Solver, 357
getNewtonMaxSteps
amici::Solver, 356
getNewtonOutput
amici::SteadystateProblem, 401
getNewtonPreequilibration
amici::Solver, 357
getNewtonSimulation
amici::SteadystateProblem, 401
getNonlinearSolverIteration
amici::Solver, 362
getNumberOfRoots
amici::ForwardProblem, 105
getParameterList
amici::Model, 165
getParameterScale
amici::Model, 160

getParameters
amici::Model, 161
getRHSAtDiscontinuities
amici::ForwardProblem, 104
getRHSBeforeDiscontinuities
amici::ForwardProblem, 105
getRelativeTolerance
amici::Solver, 358
getRelativeToleranceQuadratures
amici::Solver, 359
getReturnDataMatlabFromAmiciCall
amici, 30
getRootCounter
amici::ForwardProblem, 106
getRootIndexes
amici::ForwardProblem, 105
getRuleVars
amici::sbml_import, 69
getSensiFlag
amifun, 415
getSensis
amici::NewtonSolver, 275
getSensitivityMethod
amici::Solver, 356
getSensitivityOrder
amici::Solver, 358
getSolver
amici::Model, 121
amici::Model_DAE, 233
amici::Model_ODE, 263
amici::NewtonSolver, 273
getSparseSymLines
amici::sbml_import::SbmlImporter, 334
getSparseSymbols
amici::sbml_import::SbmlImporter, 319
getStabilityLimitFlag
amici::Solver, 364
getStateDerivativePointer
amici::ForwardProblem, 107
getStateDerivativeSensitivityPointer
amici::ForwardProblem, 108
getStateOrdering
amici::Solver, 363
getStatePointer
amici::ForwardProblem, 107
getStateSensitivity
amici::ForwardProblem, 104
getStateSensitivityPointer
amici::ForwardProblem, 108
getStatesAtDiscontinuities
amici::ForwardProblem, 104
getStep
amici::NewtonSolver, 274
getSymLines
amici::sbml_import::SbmlImporter, 333
getSymbolicDiagonal
amici::sbml_import, 68
getSymbols
amici::sbml_import, 68
getSyms
amifun, 415
getTime
amici::ForwardProblem, 103
getTimepoints
amici::Model, 163
getUnscaledParameters
amici::Model, 162
getVector
amici::AmiVector, 77
getdJydx
amici::BackwardProblem, 92
getdxBptr
amici::BackwardProblem, 91
getmy
amici::Model, 194
getmz
amici::Model, 195
getoptimized
optsym, 454
getrz
amici::Model, 199
getsrz
amici::Model, 200
getsx
amici::Model, 198
getsz
amici::Model, 200
gett
amici::BackwardProblem, 89
amici::Model, 175
getwhich
amici::BackwardProblem, 90
getwhichptr
amici::BackwardProblem, 90
getx
amici::Model, 197
getxBptr
amici::BackwardProblem, 90
getxQBptr
amici::BackwardProblem, 91
gety
amici::Model, 196
getz
amici::Model, 198

h
amici::Model, 211
HTable
amimodel, 431
heaviside
amici, 49
hflag
amievent, 409

IDAException, 110
amici::IDAException, 110
id

amimodel, 435
idlist
amici::Model, 205
init
amici::Solver, 365
initAndAttachArray
amici, 40
initHeaviside
amici::Model, 155
initMatlabDiagnosisFields
amici, 32
initMatlabReturnFields
amici, 31
initialize
amici::Model, 154
initializeLinearSolver
amici::Solver, 393
initializeLinearSolverB
amici::Solver, 395
initializeStates
amici::Model, 154
IntegrationFailure, 111
amici::IntegrationFailure, 112
interface_matlab.cpp, 470
mexFunction, 471
InternalSensitivityMethod
amici, 20
interpType
amioption, 446
InterpolationType
amici, 20
invalidate
amici::ReturnData, 290
invalidateLLH
amici::ReturnData, 292
isInf
amici, 55
isNaN
amici, 54
ism
amioption, 446
iter
amioption, 445

J
amici::Model, 207
amici::ReturnData, 294

k
amici::Model, 158

lbw
amici::Model, 204
amimodel, 436
LinearMultistepMethod
amici, 20
LinearSolver
amici, 19
linsol
amioption, 445
linsolveSPBCG
amici::NewtonSolverIterative, 284
llh
amici::ReturnData, 300
lmm
amioption, 445
loadOldHashes
amimodel, 427
loadSBMLFile
amici::sbml_import::SbmlImporter, 309
log
amici, 48

M
amici::Model, 211
M_PI
amici.cpp, 468
makeEvents
amimodel, 425
makeSyms
amimodel, 426
max
amici, 52
maxflag
amimodel, 440
maxlinsteps
amici::NewtonSolver, 277
maxsteps
amici::NewtonSolver, 277
amioption, 444
maxstepsB
amioption, 444
mexFunction
interface_matlab.cpp, 471
mfuns
amimodel, 438
min
amici, 50
minflag
amimodel, 440
minus
amici::AmiVector, 78
Model, 113
amici::Model, 118–120
model
amici::ForwardProblem, 109
amici::NewtonSolver, 277
amised, 451
Model_DAE, 214
amici::Model_DAE, 215
Model_ODE, 242
amici::Model_ODE, 244, 245
modelname
amimodel, 431
amised, 451
msgIdAndTxtFp
amici, 18
my

amici::ExpData, 98
amici::Model, 208
mz
amici::ExpData, 99
amici::Model, 208

nMaxEvent
amici::Model, 159
ndwdp
amici::Model, 204
amimodel, 441
ndwdx
amici::Model, 203
amimodel, 441
ne
amici::Model, 203
amici::ReturnData, 302
amidata, 405
nevent
amimodel, 434
newton_maxlinsteps
amioption, 448
newton_maxsteps
amici::ReturnData, 303
amioption, 448
newton_numlinsteps
amici::ReturnData, 299
newton_numsteps
amici::ReturnData, 299
newton_preeq
amioption, 448
newton_status
amici::ReturnData, 299
newton_time
amici::ReturnData, 299
NewtonFailure, 270
amici::NewtonFailure, 271
NewtonSolver, 271
amici::NewtonSolver, 273
NewtonSolverDense, 279
amici::NewtonSolverDense, 279
NewtonSolverIterative, 281
amici::NewtonSolverIterative, 282
NewtonSolverSparse, 285
amici::NewtonSolverSparse, 286
ng
amimodel, 434
nJ
amici::Model, 204
amici::ReturnData, 303
nk
amici::Model, 158
amici::ReturnData, 301
amidata, 405
amimodel, 434
nmaxevent
amici::ExpData, 100
amici::Model, 213
amici::ReturnData, 303
amioption, 447
nnz
amici::Model, 204
amimodel, 436
NonlinearSolverIteration
amici, 20
np
amici::Model, 157
amici::ReturnData, 301
amimodel, 434
nplist
amici::Model, 156
amici::ReturnData, 303
nt
amici::ExpData, 99
amici::Model, 159
amici::ReturnData, 303
amidata, 404
numerrtestfails
amici::ReturnData, 298
numerrtestfailsB
amici::ReturnData, 298
numnonlinsolvconvfails
amici::ReturnData, 298
numnonlinsolvconvfailsB
amici::ReturnData, 298
numrhsevals
amici::ReturnData, 297
numrhsevalsB
amici::ReturnData, 297
numsteps
amici::ReturnData, 297
numstepsB
amici::ReturnData, 297
nvecs
amifun, 417
nw
amici::Model, 203
amimodel, 440
nx
amici::Model, 202
amici::ReturnData, 301
amimodel, 433
nxtrue
amici::Model, 202
amici::ReturnData, 301
amimodel, 433
ny
amici::Model, 202
amici::ReturnData, 302
amidata, 404
amimodel, 433
nytrue
amici::ExpData, 99
amici::Model, 202
amici::ReturnData, 302
amimodel, 433
nz

amici::Model, 203
amici::ReturnData, 302
amidata, 404
amimodel, 435
nztrue
amici::ExpData, 99
amici::Model, 203
amici::ReturnData, 302
amimodel, 435

o2flag
amimodel, 439
o2mode
amici::Model, 205
amici::ReturnData, 304
operator=
amici::AmiVector, 75
amici::Model, 120
operator==
amici, 30, 44
amici::Model, 201
amici::Solver, 397
operator[]
amici::AmiVector, 79
amici::AmiVectorArray, 85
optsym, 453
getoptimized, 454
optsym, 453
order
amici::ReturnData, 298
ordering
amioption, 447
originalParameters
amici::Model, 212
outputcount
amised, 451

param
amimodel, 438
parseModel
amimodel, 422
pi
amici, 66
plist
amici::Model, 169
plist_
amici::Model, 212
prepareLinearSystem
amici::NewtonSolver, 275
amici::NewtonSolverDense, 280
amici::NewtonSolverIterative, 283
amici::NewtonSolverSparse, 287
prepareModelFolder
amici::sbml_import::SbmlImporter, 325
printErrMsgIdAndTxt
amici, 20
printWarnMsgIdAndTxt
amici, 21
printWithException
amici::sbml_import::SbmlImporter, 335
processCompartments
amici::sbml_import::SbmlImporter, 315
processParameters
amici::sbml_import::SbmlImporter, 314
processReactions
amici::sbml_import::SbmlImporter, 315
processRules
amici::sbml_import::SbmlImporter, 315
processSBML
amici::sbml_import::SbmlImporter, 312
processSpecies
amici::sbml_import::SbmlImporter, 314
processTime
amici::sbml_import::SbmlImporter, 316
processVolumeConversion
amici::sbml_import::SbmlImporter, 316
pscale
amici::Model, 213
amici::ReturnData, 304
amioption, 449

qbinit
amici::Solver, 366
quad_atol
amioption, 444
quad_rtol
amioption, 444

rdata
amici::ForwardProblem, 109
amici::NewtonSolver, 278
rdataToNumPyArrays
amici, 63
realtype
amici, 18
recompile
amimodel, 439
reorder
amici, 41
replaceInAllExpressions
amici::sbml_import::SbmlImporter, 317
replaceSpecialConstants
amici::sbml_import::SbmlImporter, 318
res
amici::ReturnData, 296
reset
amici::AmiVector, 78
amici::AmiVectorArray, 86
ReturnData, 288
amici::ReturnData, 290
rootInit
amici::Solver, 367
rowvals
amimodel, 436
rowvalsB
amimodel, 437
rtol
amici::NewtonSolver, 277

amioption, 443
runAmiciSimulation
amici, 21, 62
rz
amici::ReturnData, 295

s2llh
amici::ReturnData, 300
s2rz
amici::ReturnData, 295
SBML2AMICI.m, 472
SBML2AMICI, 472
SBML2AMICI
SBML2AMICI.m, 472
SBMLException, 305
sbml2amici
amici::sbml_import::SbmlImporter, 309
SbmlImporter, 305
sedml
amised, 451
sens_ind
amioption, 444
sensInit1
amici::Solver, 367
sensi
amici::ReturnData, 304
amioption, 446
sensi_meth
amici::ReturnData, 304
amioption, 446
sensiflag
amifun, 417
serializeToChar
amici, 42
serializeToStdVec
amici, 43
serializeToString
amici, 43
set
amici::AmiVector, 79
setAbsoluteTolerance
amici::Solver, 359
setAbsoluteToleranceQuadratures
amici::Solver, 360
setBandJacFn
amici::Solver, 368
setBandJacFnB
amici::Solver, 370
setDenseJacFn
amici::Solver, 368
setDenseJacFnB
amici::Solver, 369
setFixedParameters
amici::Model, 162
setHflag
amievent, 408
setInitialStateSensitivities
amici::Model, 167
setInitialStates
amici::Model, 166
setInternalSensitivityMethod
amici::Solver, 365
setInterpolationType
amici::Solver, 363
setJacTimesVecFn
amici::Solver, 369
setJacTimesVecFnB
amici::Solver, 370
setLinearMultistepMethod
amici::Solver, 362
setLinearSolver
amici::Solver, 364
setMaxSteps
amici::Solver, 361
setMaxStepsBackwardProblem
amici::Solver, 361
setModelData
amici, 26
setNMaxEvent
amici::Model, 159
setName
amici::sbml_import::SbmlImporter, 310
setNewtonMaxLinearSteps
amici::Solver, 358
setNewtonMaxSteps
amici::Solver, 356
setNewtonPreequilibration
amici::Solver, 357
setNonlinearSolverIteration
amici::Solver, 362
setObservedData
amici::ExpData, 96
setObservedDataStdDev
amici::ExpData, 97
setObservedEvents
amici::ExpData, 97
setObservedEventsStdDev
amici::ExpData, 98
setParameterList
amici::Model, 166
setParameterScale
amici::Model, 160
setParameters
amici::Model, 161
setPaths
amici::sbml_import::SbmlImporter, 311
setRelativeTolerance
amici::Solver, 359
setRelativeToleranceQuadratures
amici::Solver, 360
setSensitivityMethod
amici::Solver, 356
setSensitivityOrder
amici::Solver, 358
setSolverOptions
amici, 27
setSparseJacFn

amici::Solver, 368
setSparseJacFnB
amici::Solver, 369
setStabilityLimitFlag
amici::Solver, 364
setStateOrdering
amici::Solver, 363
setT0
amici::Model, 168
setTimepoints
amici::Model, 163
setupAMIB
amici::Solver, 344
setupAMI
amici::Solver, 342
SetupFailure, 336
amici::SetupFailure, 337
setupReturnData
amici, 27
seval
amici, 46
Sigma_Y
amidata, 406
Sigma_Z
amidata, 406
sigmay
amici::ExpData, 98
amici::Model, 205
amici::ReturnData, 296
sigmaz
amici::ExpData, 99
amici::Model, 206
amici::ReturnData, 294
sign
amici, 57
sinteg
amici, 47
sllh
amici::ReturnData, 300
solveLinearSystem
amici::NewtonSolver, 276
amici::NewtonSolverDense, 280
amici::NewtonSolverIterative, 283
amici::NewtonSolverSparse, 287
Solver, 337
amici::Solver, 341
solver
amici::ForwardProblem, 109
solverWasCalled
amici::Solver, 398
sparseidx
amimodel, 436
sparseidxB
amimodel, 437
spline
amici, 45, 57
spline.cpp, 473
spline_pos
amici, 58
splineflag
amimodel, 440
sres
amici::ReturnData, 297
srz
amici::ReturnData, 295
ss
amioption, 447
ssigmay
amici::ReturnData, 296
ssigmaz
amici::ReturnData, 294
StateOrdering
amici, 20
status
amici::ReturnData, 301
stau
amici::Model, 211
SteadystateProblem, 398
amici::SteadystateProblem, 398
stldet
amioption, 445
storeBacktrace
amici::AmiException, 72
strsym
amifun, 416
strsym_old
amifun, 416
sx
amici::ReturnData, 295
sx0
amici::ReturnData, 300
amioption, 447
sx0data
amici::Model, 213
sy
amici::ReturnData, 296
sym
amifun, 415
amimodel, 430
sym_noopt
amifun, 415
symbolic_functions.cpp, 473
sz
amici::ReturnData, 294

t
amici::Model, 164
amici::NewtonSolver, 277
amidata, 405
t0
amici::Model, 167
amimodel, 432
TemplateAmici, 336
time
amici::IntegrationFailure, 112
trigger
amievent, 409

ts
amici::Model, 213
amici::ReturnData, 293
tstart
amici::Model, 213
amioption, 445
turnOffRootFinding
amici::Solver, 355

ubw
amici::Model, 204
amimodel, 435
unscaleParameters
amici::Model, 169
unscaledParameters
amici::Model, 212
updateHeaviside
amici::Model, 174
updateHeavisideB
amici::Model, 175
updateModelName
amimodel, 422
updateRHS
amimodel, 421
updateWrapPath
amimodel, 422

varidx
amised, 452
varsym
amised, 452

w
amici::Model, 210
warnMsgIdAndTxt
amici, 66
what
amici::AmiException, 71
workBackwardProblem
amici::BackwardProblem, 88
workForwardProblem
amici::ForwardProblem, 102
workSteadyStateProblem
amici::SteadystateProblem, 399
wrap_path
amimodel, 439
wrapErrHandlerFn
amici::Solver, 371
writeCMakeFile
amici::sbml_import::SbmlImporter, 331
writeCcode
amifun, 412
writeCcode_sensi
amifun, 411
writeFunctionFile
amici::sbml_import::SbmlImporter, 327
writeIndexFiles
amici::sbml_import::SbmlImporter, 327
writeMatlabField0
amici, 34
writeMatlabField1
amici, 35
writeMatlabField2
amici, 36
writeMatlabField3
amici, 37
writeMatlabField4
amici, 38
writeMcode
amifun, 412
writeModelHeader
amici::sbml_import::SbmlImporter, 330
writeModuleSetup
amici::sbml_import::SbmlImporter, 332
writeSwigFiles
amici::sbml_import::SbmlImporter, 332
writeWrapfunctionsCPP
amici::sbml_import::SbmlImporter, 329
writeWrapfunctionsHeader
amici::sbml_import::SbmlImporter, 330
wtype
amimodel, 432

x
amici::NewtonSolver, 278
amici::ReturnData, 295
x0
amici::ReturnData, 299
amioption, 447
x0data
amici::Model, 212
xdot
amici::NewtonSolver, 278
amici::ReturnData, 294

Y
amidata, 405
y
amici::ReturnData, 296

Z
amidata, 406
z
amici::ReturnData, 294
amievent, 409
z2event
amici::Model, 205
amimodel, 440
amioption, 448