AMICI

Generated by Doxygen 1.8.14

Contents

1 About AMICI **2**

2 License Conditions **3**

3 How to contribute **3**

4 Installation **4**

5 Python Interface **6**

6 MATLAB Interface **7**

7 C++ Interface **13**

8 FAQ **14**

9 Class Documentation **15**

9.1 AmiException Class Reference 15

9.2 AmiVector Class Reference 17

9.3 AmiVectorArray Class Reference 24

9.4 BackwardProblem Class Reference 30

9.5 CvodeException Class Reference 37

9.6 ExpData Class Reference 38

9.7 ForwardProblem Class Reference 59

9.8 IDAException Class Reference 68

9.9 IntegrationFailure Class Reference 69

9.10 Model Class Reference 71

9.11 Model_DAE Class Reference 185

9.12 Model_ODE Class Reference 213

9.13 NewtonFailure Class Reference 241

9.14 NewtonSolver Class Reference 242

9.15 NewtonSolverDense Class Reference 250

9.16 NewtonSolverIterative Class Reference 252

9.17 NewtonSolverSparse Class Reference . . . 256
9.18 ReturnData Class Reference . . . 259
9.19 SetupFailure Class Reference . . . 276
9.20 Solver Class Reference . . . 277
9.21 SteadystateProblem Class Reference . . . 339
9.22 Model_model_steadystate_scaled Class Reference . . . 346
9.23 PyHeapTypeObject Struct Reference . . . 384
9.24 less< PyObject * > Struct Template Reference . . . 384
9.25 less< swig::SwigPtr_PyObject > Struct Template Reference . . . 384
9.26 less< swig::SwigVar_PyObject > Struct Template Reference . . . 384
9.27 from_oper< ValueType > Struct Template Reference . . . 385
9.28 noconst_traits< Type > Struct Template Reference . . . 385
9.29 noconst_traits< const Type > Struct Template Reference . . . 385
9.30 pointer_category Struct Reference . . . 386
9.31 stop_iteration Struct Reference . . . 386
9.32 SwigPtr_PyObject Class Reference . . . 386
9.33 SwigPyIterator Struct Reference . . . 387
9.34 SwigPyIterator_T< OutIterator > Class Template Reference . . . 388
9.35 SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper > Class Template Reference . . . 389
9.36 SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper > Class Template Reference . . . 390
9.37 SwigPySequence_ArrowProxy< T > Struct Template Reference . . . 391
9.38 SwigPySequence_Cont< T > Struct Template Reference . . . 391
9.39 SwigPySequence_InputIterator< T, Reference > Struct Template Reference . . . 392
9.40 SwigPySequence_Ref< T > Struct Template Reference . . . 393
9.41 SwigVar_PyObject Struct Reference . . . 393
9.42 traits< Type > Struct Template Reference . . . 394
9.43 traits< PyObject * > Struct Template Reference . . . 394
9.44 traits< Type * > Struct Template Reference . . . 394
9.45 traits_as< Type, Category > Struct Template Reference . . . 395
9.46 traits_as< Type *, pointer_category > Struct Template Reference . . . 395

9.47 traits_as< Type, pointer_category > Struct Template Reference 395

9.48 traits_as< Type, value_category > Struct Template Reference 396

9.49 traits_asptr< Type > Struct Template Reference 396

9.50 traits_asval< Type > Struct Template Reference 396

9.51 traits_asval< PyObject *> Struct Template Reference 396

9.52 traits_asval< Type * > Struct Template Reference 397

9.53 traits_check< Type, Category > Struct Template Reference 397

9.54 traits_check< PyObject *, value_category > Struct Template Reference 397

9.55 traits_check< Type, pointer_category > Struct Template Reference 398

9.56 traits_check< Type, value_category > Struct Template Reference 398

9.57 traits_from< Type > Struct Template Reference 398

9.58 traits_from< const Type * > Struct Template Reference 398

9.59 traits_from< PyObject * > Struct Template Reference 399

9.60 traits_from< Type * > Struct Template Reference 399

9.61 traits_from_ptr< Type > Struct Template Reference 399

9.62 traits_info< Type > Struct Template Reference 400

9.63 traits_info< Type * > Struct Template Reference 400

9.64 traits_reserve< Sequence > Struct Template Reference 400

9.65 value_category Struct Reference 401

9.66 swig_cast_info Struct Reference 401

9.67 swig_const_info Struct Reference 401

9.68 swig_globalvar Struct Reference 401

9.69 swig_module_info Struct Reference 402

9.70 swig_dycast_func Struct Reference 402

9.71 swig_varlinkobject Struct Reference 402

9.72 SwigPyClientData Struct Reference 402

9.73 SwigPyObject Struct Reference 403

9.74 SwigPyPacked Struct Reference 403

10 File Documentation 403

10.1 amici.cpp File Reference 403

10.2 cblas.cpp File Reference 406

10.3 interface_matlab.cpp File Reference 409

10.4 spline.cpp File Reference 417

10.5 symbolic_functions.cpp File Reference 420

Index 429

1 About AMICI

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

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

Publications

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

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

Current build status

2 License Conditions

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

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

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, 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] Before replacing the test results, confirm that only expected datasets have changed, e.g. using `h5diff -v -r 1e-8 tests/cpputest/expectedResults.h5 tests/cpputest/writeResults.h5.bak | less`

4 Installation

Availability

The sources for AMICI are accessible as

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

Obtaining AMICI via the GIT versioning system

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

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

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

Installation

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

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` Note that Microsoft Visual Studio compilers are currently not supported.

Python

To use AMICI from Python, install the module using pip

```
pip3 install amici
```

You can now import it as python module:

```
import amici
```

C++

To use AMICI from C++, run the

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

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

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

In CMake-based packages, amici can be linked via

```
find_package(Amici)
```

Dependencies

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

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

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

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

AMICI uses the following packages from SUNDIALS:

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

IDAS

AMICI uses the following packages from SuiteSparse:

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

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

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

5 Python Interface

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

Model Definition

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

SBML input

First, import an sbml file using the `amici.sbml_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.Sbml↩Importer.sbml2amici`, passing all the previously defined model specifications

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

Model Simulation

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

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

the compiled model can now be imported as python module

```
import test as modelModule
```

to obtain a model instance call the `getModel()` method. This model instance will be instantiated using the 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.sym.xdot(1) = [ const1 - param1*state1 ];
model.sym.xdot(2) = [ +param2*state1 + dirac(t-param3) - const2*state2 ];
model.sym.xdot(3) = [ param4*state2 ];
```

or

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

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

For DAEs also specify the mass matrix.

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

The specification of M may depend on parameters and constants.

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

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

Dirac functions can be used to cause a jump in the respective states at the specified time-point. This is typically used to model injections, or other external stimuli. Spline functions can be used to model time/state dependent response with 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.sym.x0 = [ param4, 0, 0 ];
```

Observables

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

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

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

Events

Specifying events is optional. Events are specified in terms of a trigger function, a bolus 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.sym.event(1) = amievent(state1 - state2,0,[]);
```

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

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

Standard Deviation

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

Standard deviaton for observable data is denoted by sigma_y

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

Standard deviaton for event data is denoted by sigma_t

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

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

Objective Function

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

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

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

```
model.sym.Jy
```

As reference see the default specification of `this.sym.Jy` in `amimodel.makeSyms`.

SBML

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

Model Compilation

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

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

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

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

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

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

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

Model Simulation

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

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

Integration

Define a time vector:

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

Generate a parameter vector:

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

Generate a constants vector:

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

Integrate:

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

The integration status will be indicated by the `sol.status` flag. Negative values indicated failed integration. The states will then be available as sol.x. The observables will then be available as `sol.y`. The event outputs will then be available as `sol.z`. If no event 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 can also be called via

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

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

Forward Sensitivities

Set the sensitivity computation to forward sensitivities and integrate:

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

The integration status will be indicated by the `sol.status` flag. Negative values indicate failed integration. The states will be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will be available as `sol.y`, with the derivative with respect to the parameters in `sol.sy`. The event outputs will be available as `sol.z`, with the derivative with respect to the parameters in `sol.sz`. If no event 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 can also be called via

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

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

Adjoint Sensitivities

Set the sensitivity computation to adjoint sensitivities:

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

Define Experimental Data:

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

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

Integrate:

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

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

Steady State Sensitivities

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

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

Set the final timepoint as infinity, this will indicate the solver to compute the 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, or use the CMake package configuration from the build directory which tells CMake about all AMICI dependencies.

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

8 FAQ

Q: My model fails to build.

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

Q: It still does not compile.

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

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

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

Q: The simulation/sensitivities I get are incorrect.

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

9 Class Documentation

9.1 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)

9.1.1 Detailed Description

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

Definition at line 29 of file exception.h.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 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:

9.1.2.2 AmiException() [2/2]

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

copy constructor

Parameters

old	object to copy from

Definition at line 43 of file exception.h.

9.1.3 Member Function Documentation

9.1.3.1 what()

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

override of default error message function

Returns

msg error message

Definition at line 54 of file exception.h.

9.1.3.2 getBacktrace()

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

returns the stored backtrace

Returns

trace backtrace

Definition at line 61 of file exception.h.

9.1.3.3 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:

9.2 AmiVector Class Reference

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

Public Member Functions

- AmiVector (const long int length)
- AmiVector (std::vector< realtype > rvec)
- AmiVector (const AmiVector &vold)
- AmiVector & operator= (AmiVector &other)
- realtype * data ()
- const realtype * data () const

- N_Vector getNVector () const
- std::vector< realtype > const & getVector () const
- int getLength () const
- void reset ()
- void minus ()
- void set (realtype val)
- realtype & operator[ ] (int pos)
- realtype & at (int pos)

9.2.1 Detailed Description

AmiVector class provides a generic interface to the NVector_Serial struct

Definition at line 11 of file include/amici/vector.h.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 AmiVector() [1/3]

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

Creates an std::vector<realtype> and attaches the data pointer to a newly created N_Vector_Serial. Using N↩_VMake_Serial ensures that the N_Vector module does not try to deallocate the data vector when calling N_V↩Destroy_Serial

Parameters

length	number of elements in vector

Returns

new AmiVector instance

Definition at line 23 of file include/amici/vector.h.

9.2.2.2 AmiVector() [2/3]

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

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

Parameters

rvec	vector from which the data will be copied

Returns

new AmiVector instance

Definition at line 34 of file include/amici/vector.h.

9.2.2.3 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 include/amici/vector.h.

9.2.3 Member Function Documentation

9.2.3.1 operator=()

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

copy-move assignment operator

Parameters

other	right hand side

Returns

left hand side

Definition at line 52 of file include/amici/vector.h.

9.2.3.2 data() [1/2]

```
realtype* data ( )
```

data accessor

Returns

pointer to data array

Definition at line 63 of file include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.3 data() [2/2]

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

const data accessor

Returns

const pointer to data array

Definition at line 70 of file include/amici/vector.h.

9.2.3.4 getNVector()

```
N_Vector getNVector ( ) const
```

N_Vector accessor

Returns

N_Vector

Definition at line 77 of file include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.5 getVector()

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

Vector accessor

Returns

Vector

Definition at line 84 of file include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.6 getLength()

```
int getLength ( ) const
```

returns the length of the vector

Returns

length

Definition at line 91 of file include/amici/vector.h.

9.2.3.7 reset()

```
void reset ( )
```

resets the Vector by filling with zero values

Definition at line 97 of file include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.8 minus()

```
void minus ( )
```

changes the sign of data elements

Definition at line 103 of file include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.9 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 include/amici/vector.h.

Here is the caller graph for this function:

9.2.3.10 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 include/amici/vector.h.

9.2.3.11 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 include/amici/vector.h.

Here is the caller graph for this function:

9.3 AmiVectorArray Class Reference

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

Public Member Functions

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

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

9.3.1 Detailed Description

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

Definition at line 148 of file include/amici/vector.h.

9.3.2 Constructor & Destructor Documentation

9.3.2.1 AmiVectorArray() [1/2]

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

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

Parameters

length_inner	length of vectors
length_outer	number of vectors

Returns

New AmiVectorArray instance

Definition at line 159 of file include/amici/vector.h.

9.3.2.2 AmiVectorArray() [2/2]

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

copy constructor

Parameters

vaold	object to copy from

Returns

new AmiVectorArray instance

Definition at line 173 of file include/amici/vector.h.

Here is the call graph for this function:

9.3.3 Member Function Documentation

9.3.3.1 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 184 of file include/amici/vector.h.

Here is the caller graph for this function:

9.3.3.2 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 192 of file include/amici/vector.h.

9.3.3.3 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 201 of file include/amici/vector.h.

Here is the caller graph for this function:

9.3.3.4 getNVectorArray()

```
N_Vector* getNVectorArray ( )
```

accessor to NVectorArray

Returns

N_VectorArray

Definition at line 208 of file include/amici/vector.h.

9.3.3.5 getNVector()

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

accessor to NVector element

Parameters

pos	index of corresponding AmiVector

Returns

N_Vector

Definition at line 216 of file include/amici/vector.h.

9.3.3.6 operator[]() [1/2]

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

accessor to AmiVector elements

Parameters

pos	index of AmiVector

Returns

AmiVector

Definition at line 224 of file include/amici/vector.h.

9.3.3.7 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 232 of file include/amici/vector.h.

9.3.3.8 getLength()

```
int getLength ( ) const
```

length of AmiVectorArray

Returns

length

Definition at line 239 of file include/amici/vector.h.

Here is the caller graph for this function:

9.3.3.9 reset()

```
void reset ( )
```

resets every AmiVector in AmiVectorArray

Definition at line 244 of file include/amici/vector.h.

Here is the caller graph for this function:

9.4 BackwardProblem Class Reference

class to solve backwards problems.

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

Public Member Functions

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

9.4.1 Detailed Description

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

Definition at line 23 of file backwardproblem.h.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 BackwardProblem()

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

Construct backward problem from forward problem

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:

9.4.3 Member Function Documentation

9.4.3.1 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 42 of file backwardproblem.cpp.

Here is the call graph for this function:

9.4.3.2 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:

9.4.3.3 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:

9.4.3.4 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:

9.4.3.5 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:

9.4.3.6 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:

9.4.3.7 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:

9.4.3.8 getdJydx()

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

accessor for dJydx

Returns

dJydx

Definition at line 74 of file backwardproblem.h.

Here is the caller graph for this function:

9.5 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)

9.5.1 Detailed Description

Definition at line 117 of file exception.h.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 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.

9.6 ExpData Class Reference

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

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

Public Member Functions

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

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

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

  ExpData.
- ExpData (const Model &model)
- ExpData (const ReturnData &rdata, realtype sigma_y, realtype sigma_z)
- ExpData (const ReturnData &rdata, std::vector< realtype > sigma_y, std::vector< realtype > sigma_z)
- ExpData (const ExpData &other)

  Copy constructor.
- const int nt () const
- void setTimepoints (const std::vector< realtype > &ts)
- std::vector< realtype > const & getTimepoints () const
- realtype getTimepoint (int it) const
- void setObservedData (const std::vector< realtype > &observedData)
- void setObservedData (const std::vector< realtype > &observedData, int iy)
- bool isSetObservedData (int it, int iy) const
- std::vector< realtype > const & getObservedData () const
- const realtype * getObservedDataPtr (int it) const
- void setObservedDataStdDev (const std::vector< realtype > &observedDataStdDev)
- void setObservedDataStdDev (const realtype stdDev)
- void setObservedDataStdDev (const std::vector< realtype > &observedDataStdDev, int iy)
- void setObservedDataStdDev (const realtype stdDev, int iy)
- bool isSetObservedDataStdDev (int it, int iy) const
- std::vector< realtype > const & getObservedDataStdDev () const
- const realtype * getObservedDataStdDevPtr (int it) const
- void setObservedEvents (const std::vector< realtype > &observedEvents)
- void setObservedEvents (const std::vector< realtype > &observedEvents, int iz)
- bool isSetObservedEvents (int ie, int iz) const
- std::vector< realtype > const & getObservedEvents () const
- const realtype * getObservedEventsPtr (int ie) const
- void setObservedEventsStdDev (const std::vector< realtype > &observedEventsStdDev)
- void setObservedEventsStdDev (const realtype stdDev)
- void setObservedEventsStdDev (const std::vector< realtype > &observedEventsStdDev, int iz)
- void setObservedEventsStdDev (const realtype stdDev, int iz)
- bool isSetObservedEventsStdDev (int ie, int iz) const
- std::vector< realtype > const & getObservedEventsStdDev () const
- const realtype * getObservedEventsStdDevPtr (int ie) const

Public Attributes

- const int nytrue
- const int nztrue
- const int nmaxevent
- std::vector< realtype > fixedParameters
- std::vector< realtype > fixedParametersPreequilibration

Protected Attributes

- std::vector< realtype > ts
- std::vector< realtype > observedData
- std::vector< realtype > observedDataStdDev
- std::vector< realtype > observedEvents
- std::vector< realtype > observedEventsStdDev

9.6.1 Detailed Description

Definition at line 14 of file edata.h.

9.6.2 Constructor & Destructor Documentation

9.6.2.1 ExpData() [1/8]

```
ExpData ( )
```

default constructor

Definition at line 13 of file edata.cpp.

9.6.2.2 ExpData() [2/8]

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

Parameters

nytrue	
nztrue	
nmaxevent	

Definition at line 15 of file edata.cpp.

9.6.2.3 ExpData() [3/8]

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

Parameters

nytrue	
nztrue	
nmaxevent	
ts	

Definition at line 20 of file edata.cpp.

Here is the call graph for this function:

9.6.2.4 ExpData() [4/8]

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

Parameters

nytrue	
nztrue	
nmaxevent	
ts	
my	
sigmay	
mz	
sigmaz	

Definition at line 27 of file edata.cpp.

9.6.2.5 ExpData() [5/8]

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

constructor that initializes with Model

Parameters

model	pointer to model specification object

Definition at line 37 of file edata.cpp.

Here is the call graph for this function:

9.6.2.6 ExpData() [6/8]

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

constructor that initializes with returnData, adds

Parameters

rdata	return data pointer with stored simulation results
sigma↩ _y	scalar standard deviations for all observables
sigma↩ _z	scalar standard deviations for all event observables

Definition at line 51 of file edata.cpp.

9.6.2.7 ExpData() [7/8]

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

constructor that initializes with returnData, adds

Parameters

rdata	return data pointer with stored simulation results
sigma↩ _y	vector of standard deviations for observables (dimension: nytrue or nytrue x nt, row-major)
sigma↩ _z	vector of standard deviations for event observables (dimension: nztrue or nztrue x nmaxevent, row-major)

Definition at line 56 of file edata.cpp.

Here is the call graph for this function:

9.6.2.8 ExpData() [8/8]

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

Parameters

other	object to copy from

Definition at line 43 of file edata.cpp.

9.6.3 Member Function Documentation

9.6.3.1 nt()

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

number of timepoints

Returns

number of timepoints

Definition at line 115 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.2 setTimepoints()

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

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

Parameters

ts	timepoints

Definition at line 105 of file edata.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.6.3.3 getTimepoints()

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

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

Returns

ExpData::ts

Definition at line 111 of file edata.cpp.

9.6.3.4 getTimepoint()

```
realtype getTimepoint (
            int it ) const
```

get function that returns timepoint at index

Parameters

it	timepoint index

Returns

timepoint timepoint at index

Definition at line 119 of file edata.cpp.

9.6.3.5 setObservedData() [1/2]

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

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

Parameters

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

Definition at line 123 of file edata.cpp.

Here is the call graph for this function:

9.6.3.6 setObservedData() [2/2]

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

set function that copies observed data for specific observable

Parameters

observedData	observed data (dimension: nt)
iy	oberved data index

Definition at line 133 of file edata.cpp.

Here is the call graph for this function:

9.6.3.7 isSetObservedData()

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

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

Parameters

it	time index
iy	observable index

Returns

boolean specifying if data was set

Definition at line 141 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.8 getObservedData()

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

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

Returns

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

Definition at line 145 of file edata.cpp.

9.6.3.9 getObservedDataPtr()

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

get function that returns a pointer to observed data at index

Parameters

it	timepoint index

Returns

pointer to observed data at index (dimension: nytrue)

Definition at line 149 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.10 setObservedDataStdDev() [1/4]

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

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

Parameters

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

Definition at line 156 of file edata.cpp.

Here is the call graph for this function:

9.6.3.11 setObservedDataStdDev() [2/4]

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

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

Parameters

stdDev	standard deviation (dimension: scalar)

Definition at line 166 of file edata.cpp.

9.6.3.12 setObservedDataStdDev() [3/4]

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

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

Parameters

observedDataStdDev	standard deviation of observed data (dimension: nt)
iy	observed data index

Definition at line 170 of file edata.cpp.

Here is the call graph for this function:

9.6.3.13 setObservedDataStdDev() [4/4]

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

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

Parameters

stdDev	standard deviation (dimension: scalar)
iy	observed data index

Definition at line 178 of file edata.cpp.

Here is the call graph for this function:

9.6.3.14 isSetObservedDataStdDev()

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

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

Parameters

it	time index
iy	observable index

Returns

boolean specifying if standard deviation of data was set

Definition at line 183 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.15 getObservedDataStdDev()

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

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

Returns

standard deviation of observed data

Definition at line 187 of file edata.cpp.

9.6.3.16 getObservedDataStdDevPtr()

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

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

Parameters

it	timepoint index

Returns

pointer to standard deviation of observed data at index

Definition at line 191 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.17 setObservedEvents() [1/2]

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

set function that copies observed event data from input to ExpData::observedEvents

Parameters

observedEvents	observed data (dimension: nmaxevent x nztrue, row-major)

Definition at line 198 of file edata.cpp.

9.6.3.18 setObservedEvents() [2/2]

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

set function that copies observed event data for specific event observable

Parameters

observedEvents	observed data (dimension: nmaxevent)
iz	observed event data index

Definition at line 208 of file edata.cpp.

9.6.3.19 isSetObservedEvents()

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

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

Parameters

ie	event index
iz	event observable index

Returns

boolean specifying if data was set

Definition at line 217 of file edata.cpp.

9.6.3.20 getObservedEvents()

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

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

Returns

observed event data

Definition at line 221 of file edata.cpp.

9.6.3.21 getObservedEventsPtr()

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

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

Parameters

ie	event occurence

Returns

pointer to observed event data at ieth occurence

Definition at line 225 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.22 setObservedEventsStdDev() [1/4]

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

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

Parameters

observedEventsStdDev	standard deviation of observed event data

Definition at line 232 of file edata.cpp.

9.6.3.23 setObservedEventsStdDev() [2/4]

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

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

Parameters

stdDev	standard deviation (dimension: scalar)

Definition at line 242 of file edata.cpp.

9.6.3.24 setObservedEventsStdDev() [3/4]

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

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

Parameters

observedEventsStdDev	standard deviation of observed data (dimension: nmaxevent)
iz	observed data index

Definition at line 246 of file edata.cpp.

9.6.3.25 setObservedEventsStdDev() [4/4]

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

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

Parameters

stdDev	standard deviation (dimension: scalar)
iz	observed data index

Definition at line 254 of file edata.cpp.

9.6.3.26 isSetObservedEventsStdDev()

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

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

Parameters

ie	event index
iz	event observable index

Returns

boolean specifying if standard deviation of event data was set

Definition at line 259 of file edata.cpp.

Here is the caller graph for this function:

9.6.3.27 getObservedEventsStdDev()

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

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

Returns

standard deviation of observed event data

Definition at line 266 of file edata.cpp.

9.6.3.28 getObservedEventsStdDevPtr()

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

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

Parameters

ie	event occurence

Returns

pointer to standard deviation of observed event data at ieth occurence

Definition at line 270 of file edata.cpp.

Here is the caller graph for this function:

9.6.4 Member Data Documentation

9.6.4.1 nytrue

```
const int nytrue
```

number of observables

Definition at line 307 of file edata.h.

9.6.4.2 nztrue

```
const int nztrue
```

number of event observables

Definition at line 309 of file edata.h.

9.6.4.3 nmaxevent

```
const int nmaxevent
```

maximal number of event occurences

Definition at line 311 of file edata.h.

9.6.4.4 fixedParameters

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

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

Definition at line 314 of file edata.h.

9.6.4.5 fixedParametersPreequilibration

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

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

Definition at line 317 of file edata.h.

9.6.4.6 ts

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

observation timepoints (dimension: nt)

Definition at line 321 of file edata.h.

9.6.4.7 observedData

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

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

Definition at line 324 of file edata.h.

9.6.4.8 observedDataStdDev

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

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

Definition at line 326 of file edata.h.

9.6.4.9 observedEvents

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

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

Definition at line 329 of file edata.h.

9.6.4.10 observedEventsStdDev

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

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

Definition at line 332 of file edata.h.

9.7 ForwardProblem Class Reference

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

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

Public Member Functions

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

Public Attributes

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

9.7.1 Detailed Description

Definition at line 23 of file forwardproblem.h.

9.7.2 Constructor & Destructor Documentation

9.7.2.1 ForwardProblem()

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

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 42 of file forwardproblem.cpp.

9.7.3 Member Function Documentation

9.7.3.1 workForwardProblem()

```
void workForwardProblem ( )
```

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

Definition at line 77 of file forwardproblem.cpp.

Here is the call graph for this function:

9.7.3.2 getTime()

```
realtype getTime ( ) const
```

accessor for t

Returns

t

Definition at line 37 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.3.3 getStateSensitivity()

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

accessor for sx

Returns

sx

Definition at line 44 of file forwardproblem.h.

9.7.3.4 getStatesAtDiscontinuities()

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

accessor for x_disc

Returns

x_disc

Definition at line 51 of file forwardproblem.h.

9.7.3.5 getRHSAtDiscontinuities()

```
AmiVectorArray const& getRHSAtDiscontinuities ( ) const
```

accessor for xdot_disc

Returns

xdot_disc

Definition at line 58 of file forwardproblem.h.

9.7.3.6 getRHSBeforeDiscontinuities()

```
AmiVectorArray const& getRHSBeforeDiscontinuities ( ) const
```

accessor for xdot_old_disc

Returns

xdot_old_disc

Definition at line 65 of file forwardproblem.h.

9.7.3.7 getNumberOfRoots()

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

accessor for nroots

Returns

nroots

Definition at line 72 of file forwardproblem.h.

9.7.3.8 getDiscontinuities()

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

accessor for discs

Returns

discs

Definition at line 79 of file forwardproblem.h.

9.7.3.9 getRootIndexes()

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

accessor for rootidx

Returns

rootidx

Definition at line 86 of file forwardproblem.h.

9.7.3.10 getDJydx()

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

accessor for dJydx

Returns

dJydx

Definition at line 93 of file forwardproblem.h.

9.7.3.11 getDJzdx()

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

accessor for dJzdx

Returns

dJzdx

Definition at line 100 of file forwardproblem.h.

9.7.3.12 getRootCounter()

```
int getRootCounter ( ) const
```

accessor for iroot

Returns

iroot

Definition at line 107 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.3.13 getStatePointer()

```
AmiVector* getStatePointer ( )
```

accessor for pointer to x

Returns

&x

Definition at line 114 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.3.14 getStateDerivativePointer()

```
AmiVector* getStateDerivativePointer ( )
```

accessor for pointer to dx

Returns

&dx

Definition at line 121 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.3.15 getStateSensitivityPointer()

```
AmiVectorArray* getStateSensitivityPointer ( )
```

accessor for pointer to sx

Returns

&sx

Definition at line 128 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.3.16 getStateDerivativeSensitivityPointer()

```
AmiVectorArray* getStateDerivativeSensitivityPointer ( )
```

accessor for pointer to sdx

Returns

&sdx

Definition at line 135 of file forwardproblem.h.

Here is the caller graph for this function:

9.7.4 Member Data Documentation

9.7.4.1 model

```
Model* model
```

pointer to model instance

Definition at line 140 of file forwardproblem.h.

9.7.4.2 rdata

```
ReturnData* rdata
```

pointer to return data instance

Definition at line 142 of file forwardproblem.h.

9.7.4.3 solver

```
Solver* solver
```

pointer to solver instance

Definition at line 144 of file forwardproblem.h.

9.7.4.4 edata

```
const ExpData* edata
```

pointer to experimental data instance

Definition at line 146 of file forwardproblem.h.

9.8 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)

9.8.1 Detailed Description

Definition at line 129 of file exception.h.

9.8.2 Constructor & Destructor Documentation

9.8.2.1 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.

9.9 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

9.9.1 Detailed Description

Definition at line 144 of file exception.h.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 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.

9.9.3 Member Data Documentation

9.9.3.1 error_code

```
int error_code
```

error code returned by cvode/ida

Definition at line 147 of file exception.h.

9.9.3.2 time

```
realtype time
```

time of integration failure

Definition at line 149 of file exception.h.

9.10 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, amici::AMICI_o2mode o2mode, const std::vector< amici::realtype > &p, std::vector< amici::realtype > k, const std::vector< int > &plist, std::vector< amici::realtype > idlist, std::vector< int > z2event)
- Model (Model const &other)
- virtual ~Model ()
- Model & operator= (Model const &other)=delete
- virtual Model * clone () const =0

  Clone this instance.
- virtual std::unique_ptr< Solver > getSolver ()=0
- virtual void froot (realtype t, AmiVector *x, AmiVector *dx, realtype *root)=0
- virtual void fxdot (realtype t, AmiVector *x, AmiVector *dx, AmiVector *xdot)=0
- virtual void fJ (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, DlsMat J)=0
- virtual void fJSparse (realtype t, realtype cj, AmiVector *x, AmiVector *dx, AmiVector *xdot, SlsMat J)=0
- virtual void fJDiag (realtype t, AmiVector *Jdiag, realtype cj, AmiVector *x, AmiVector *dx)=0
- virtual void 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 fsigmay (const int it, ReturnData ∗rdata, const ExpData ∗edata)
- void fdsigmaydp (const int it, ReturnData ∗rdata, const ExpData ∗edata)
- void fsigmaz (const realtype t, const int ie, const int ∗nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fdsigmazdp (const realtype t, const int ie, const int ∗nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fJy (const int it, ReturnData ∗rdata, const ExpData ∗edata)
- void fJz (const int nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fJrz (const int nroots, ReturnData ∗rdata, const ExpData ∗edata)
- void fdJydy (const int it, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJydsigma (const int it, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJzdz (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJzdsigma (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJrzdz (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fdJrzdsigma (const int nroots, const ReturnData ∗rdata, const ExpData ∗edata)
- void fsy (const int it, ReturnData ∗rdata)
- void fsz_tf (const int ∗nroots, const int ie, ReturnData ∗rdata)
- void fsJy (const int it, const std::vector< realtype > &dJydx, ReturnData ∗rdata)
- void fdJydp (const int it, const ExpData ∗edata, ReturnData ∗rdata)
- void fdJydx (std::vector< realtype > ∗dJydx, const int it, const ExpData ∗edata, const ReturnData ∗rdata)
- void fsJz (const int nroots, const std::vector< realtype > &dJzdx, AmiVectorArray ∗sx, ReturnData ∗rdata)
- void fdJzdp (const int nroots, realtype t, const ExpData ∗edata, const ReturnData ∗rdata)
- void fdJzdx (std::vector< realtype > ∗dJzdx, const int nroots, realtype t, const ExpData ∗edata, const ReturnData ∗rdata)
- void initialize (AmiVector ∗x, AmiVector ∗dx)
- void initializeStates (AmiVector ∗x)
- void initHeaviside (AmiVector ∗x, AmiVector ∗dx)
- int nplist () const
- 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 > const & getParameterScale () const

  getParameterScale
- void setParameterScale (AMICI_parameter_scaling pscale)

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

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

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

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

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

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

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

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

  setTimepoints
- double t (int idx) const

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

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

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

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

  setInitialStates
- std::vector< realtype > const & 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 fres (const int it, ReturnData *rdata, const ExpData *edata)

- void fchi2 (const int it, ReturnData *rdata)
- void fsres (const int it, ReturnData *rdata, const ExpData *edata)
- void fFIM (const int it, ReturnData *rdata)
- void updateHeaviside (const std::vector< int > &rootsfound)
- void updateHeavisideB (const int *rootsfound)
- realtype gett (const int it, const ReturnData *rdata) const
- int checkFinite (const int N, const realtype *array, const char *fun) const

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

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

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

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

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

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

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

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

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

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

  Get ids of the model parameters.
- virtual bool hasStateIds () const

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

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

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

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

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

  Get ids of the observables.

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

- void initializeVectors ()

  Set the nplist-dependent vectors to their proper sizes.
- virtual void fx0 (realtype *x0, const realtype t, const realtype *p, const realtype *k)
- virtual void fsx0 (realtype *sx0, const realtype t, const realtype *x0, const realtype *p, const realtype *k, const int ip)
- virtual void fstau (realtype *stau, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip, const int ie)
- virtual void fy (realtype *y, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdydp (realtype *dydp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void fdydx (realtype *dydx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fz (realtype *z, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fsz (realtype *sz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void frz (realtype *rz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fsrz (realtype *srz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *sx, const int ip)
- virtual void fdzdp (realtype *dzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void fdzdx (realtype *dzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdrzdp (realtype *drzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip)
- virtual void fdrzdx (realtype *drzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdeltax (realtype *deltax, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old)

- virtual void fdeltasx (realtype *deltasx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *sx, const realtype *stau)
- virtual void fdeltaxB (realtype *deltaxB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void fdeltaqB (realtype *deltaqB, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip, const int ie, const realtype *xdot, const realtype *xdot_old, const realtype *xB)
- virtual void fsigmay (realtype *sigmay, const realtype t, const realtype *p, const realtype *k)
- virtual void fdsigmaydp (realtype *dsigmaydp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void fsigmaz (realtype *sigmaz, const realtype t, const realtype *p, const realtype *k)
- virtual void fdsigmazdp (realtype *dsigmazdp, const realtype t, const realtype *p, const realtype *k, const int ip)
- virtual void fJy (realtype *nllh, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fJz (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fJrz (realtype *nllh, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz)
- virtual void fdJydy (realtype *dJydy, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fdJydsigma (realtype *dJydsigma, const int iy, const realtype *p, const realtype *k, const realtype *y, const realtype *sigmay, const realtype *my)
- virtual void fdJzdz (realtype *dJzdz, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fdJzdsigma (realtype *dJzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *z, const realtype *sigmaz, const realtype *mz)
- virtual void fdJrzdz (realtype *dJrzdz, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)
- virtual void fdJrzdsigma (realtype *dJrzdsigma, const int iz, const realtype *p, const realtype *k, const realtype *rz, const realtype *sigmaz)
- virtual void fw (realtype *w, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h)
- virtual void fdwdp (realtype *dwdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w)
- virtual void fdwdx (realtype *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.

9.10.1 Detailed Description

Definition at line 40 of file model.h.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 Model() [1/3]

```
Model ( )
```

default constructor

Definition at line 43 of file model.h.

9.10.2.2 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,
            amici::AMICI_o2mode o2mode,
            const std::vector< amici::realtype > & p,
            std::vector< amici::realtype > k,
            const std::vector< int > & plist,
            std::vector< amici::realtype > idlist,
            std::vector< int > z2event )
```

constructor with model dimensions

Parameters

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 451 of file model.cpp.

9.10.2.3 Model() [3/3]

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

Copy constructor

Parameters

other	object to copy from

Returns

Definition at line 525 of file model.cpp.

9.10.2.4 ~Model()

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

destructor

Definition at line 583 of file model.cpp.

9.10.3 Member Function Documentation

9.10.3.1 operator=()

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

Copy assignment is disabled until const members are removed

Parameters

other	object to copy from

Returns

9.10.3.2 clone()

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

Returns

The clone

Implemented in Model_model_steadystate_scaled.

9.10.3.3 getSolver()

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

Retrieves the solver object

Returns

The Solver instance

Implemented in Model_ODE, and Model_DAE.

9.10.3.4 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:

9.10.3.5 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.

Here is the caller graph for this function:

9.10.3.6 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:

9.10.3.7 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:

9.10.3.8 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:

9.10.3.9 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:

9.10.3.10 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:

9.10.3.11 fx0() [1/2]

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

Initial states

Parameters

x	pointer to state variables

Definition at line 606 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.12 fdx0()

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

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

Parameters

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

Definition at line 611 of file model.cpp.

Here is the caller graph for this function:

9.10.3.13 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 613 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.14 fsdx0()

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

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

Definition at line 619 of file model.cpp.

Here is the caller graph for this function:

9.10.3.15 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 627 of file model.cpp.

Here is the call graph for this function:

9.10.3.16 fy() [1/2]

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

Observables / measurements

Parameters

it	timepoint index
rdata	pointer to return data instance

Definition at line 638 of file model.cpp.

Here is the call graph for this function:

9.10.3.17 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 646 of file model.cpp.

Here is the call graph for this function:

9.10.3.18 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 665 of file model.cpp.

Here is the call graph for this function:

9.10.3.19 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 677 of file model.cpp.

Here is the call graph for this function:

9.10.3.20 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 689 of file model.cpp.

Here is the call graph for this function:

9.10.3.21 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 703 of file model.cpp.

Here is the call graph for this function:

9.10.3.22 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 715 of file model.cpp.

Here is the call graph for this function:

9.10.3.23 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 726 of file model.cpp.

Here is the call graph for this function:

9.10.3.24 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 738 of file model.cpp.

Here is the call graph for this function:

9.10.3.25 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 748 of file model.cpp.

Here is the call graph for this function:

9.10.3.26 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 760 of file model.cpp.

Here is the call graph for this function:

9.10.3.27 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 772 of file model.cpp.

Here is the call graph for this function:

9.10.3.28 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 786 of file model.cpp.

Here is the call graph for this function:

9.10.3.29 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 803 of file model.cpp.

Here is the call graph for this function:

9.10.3.30 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 817 of file model.cpp.

Here is the call graph for this function:

9.10.3.31 fsigmay() [1/2]

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

Standard deviation of measurements

Parameters

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

Definition at line 830 of file model.cpp.

Here is the call graph for this function:

9.10.3.32 fdsigmaydp() [1/2]

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

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

Parameters

it	timepoint index
rdata	pointer to return data instance
edata	pointer to ExpData data instance holding sigma values

Definition at line 851 of file model.cpp.

Here is the call graph for this function:

9.10.3.33 fsigmaz() [1/2]

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

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 884 of file model.cpp.

Here is the call graph for this function:

9.10.3.34 fdsigmazdp() [1/2]

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

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

Parameters

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

Definition at line 909 of file model.cpp.

Here is the call graph for this function:

9.10.3.35 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 939 of file model.cpp.

Here is the call graph for this function:

9.10.3.36 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 956 of file model.cpp.

Here is the call graph for this function:

9.10.3.37 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 974 of file model.cpp.

Here is the call graph for this function:

9.10.3.38 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 991 of file model.cpp.

Here is the call graph for this function:

9.10.3.39 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 1017 of file model.cpp.

Here is the call graph for this function:

9.10.3.40 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 1042 of file model.cpp.

Here is the call graph for this function:

9.10.3.41 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 1059 of file model.cpp.

Here is the call graph for this function:

9.10.3.42 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 1075 of file model.cpp.

Here is the call graph for this function:

9.10.3.43 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 1092 of file model.cpp.

Here is the call graph for this function:

9.10.3.44 fsy()

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

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

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:

9.10.3.45 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 34 of file model.cpp.

Here is the call graph for this function:

9.10.3.46 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 48 of file model.cpp.

Here is the call graph for this function:

9.10.3.47 fdJydp()

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

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

Parameters

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

Definition at line 83 of file model.cpp.

Here is the call graph for this function:

9.10.3.48 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 138 of file model.cpp.

Here is the call graph for this function:

9.10.3.49 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 164 of file model.cpp.

Here is the call graph for this function:

9.10.3.50 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 205 of file model.cpp.

Here is the call graph for this function:

9.10.3.51 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 249 of file model.cpp.

Here is the call graph for this function:

9.10.3.52 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 276 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.53 initializeStates()

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

initialization of initial states

Parameters

x	pointer to state variables

Definition at line 289 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.54 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 306 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.55 nplist()

```
int nplist ( ) const
```

number of paramaeters wrt to which sensitivities are computed

Returns

length of sensitivity index vector

Definition at line 323 of file model.cpp.

Here is the caller graph for this function:

9.10.3.56 np()

```
int np ( ) const
```

total number of model parameters

Returns

length of parameter vector

Definition at line 327 of file model.cpp.

Here is the caller graph for this function:

9.10.3.57 nk()

```
int nk ( ) const
```

number of constants

Returns

length of constant vector

Definition at line 331 of file model.cpp.

Here is the caller graph for this function:

9.10.3.58 k()

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

constants

Returns

pointer to constants array

Definition at line 335 of file model.cpp.

Here is the caller graph for this function:

9.10.3.59 nMaxEvent()

```
int nMaxEvent ( ) const
```

Returns

nmaxevent

Definition at line 339 of file model.cpp.

9.10.3.60 setNMaxEvent()

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

Parameters

nmaxevent	

Definition at line 343 of file model.cpp.

9.10.3.61 nt()

```
int nt ( ) const
```

Returns

number of timepoints

Definition at line 347 of file model.cpp.

Here is the caller graph for this function:

9.10.3.62 getParameterScale()

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

Returns

Definition at line 351 of file model.cpp.

9.10.3.63 setParameterScale() [1/2]

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

Parameters

pscale	

Definition at line 355 of file model.cpp.

Here is the call graph for this function:

9.10.3.64 setParameterScale() [2/2]

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

Parameters

pscale	

Definition at line 361 of file model.cpp.

Here is the call graph for this function:

9.10.3.65 getParameters()

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

Returns

The user-set parameters (see also getUnscaledParameters)

Definition at line 367 of file model.cpp.

9.10.3.66 setParameters()

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

Parameters

p	

Definition at line 371 of file model.cpp.

Here is the call graph for this function:

9.10.3.67 getUnscaledParameters()

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

Returns

The unscaled parameters

Definition at line 379 of file model.cpp.

Here is the caller graph for this function:

9.10.3.68 getFixedParameters()

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

Returns

Definition at line 383 of file model.cpp.

Here is the caller graph for this function:

9.10.3.69 setFixedParameters()

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

Parameters

k	

Definition at line 387 of file model.cpp.

Here is the call graph for this function:

9.10.3.70 getTimepoints()

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

Returns

Definition at line 393 of file model.cpp.

Here is the caller graph for this function:

9.10.3.71 setTimepoints()

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

Parameters

ts	

Definition at line 397 of file model.cpp.

9.10.3.72 t()

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

Parameters

idx	

Returns

Definition at line 401 of file model.cpp.

Here is the caller graph for this function:

9.10.3.73 getParameterList()

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

Returns

Definition at line 405 of file model.cpp.

Here is the caller graph for this function:

9.10.3.74 setParameterList()

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

Parameters

plist	

Definition at line 409 of file model.cpp.

Here is the call graph for this function:

9.10.3.75 getInitialStates()

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

Returns

Definition at line 418 of file model.cpp.

9.10.3.76 setInitialStates()

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

Parameters

x0	

Definition at line 422 of file model.cpp.

9.10.3.77 getInitialStateSensitivities()

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

Returns

Definition at line 428 of file model.cpp.

Here is the caller graph for this function:

9.10.3.78 setInitialStateSensitivities()

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

Parameters

sx0	

Definition at line 432 of file model.cpp.

Here is the call graph for this function:

9.10.3.79 t0()

```
double t0 ( ) const
```

initial timepoint

Returns

timepoint

Definition at line 438 of file model.cpp.

Here is the caller graph for this function:

9.10.3.80 setT0()

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

Parameters

t0	

Definition at line 442 of file model.cpp.

Here is the call graph for this function:

9.10.3.81 plist()

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

entry in parameter list

Parameters

pos	index

Returns

entry

Definition at line 446 of file model.cpp.

Here is the caller graph for this function:

9.10.3.82 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 1316 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.83 fw() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 1107 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.84 fdwdp() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 1117 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.85 fdwdx() [1/2]

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

Parameters

t	timepoint
x	Vector with the states

Definition at line 1128 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.86 fres()

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

residual function

Parameters

it	time index
rdata	ReturnData instance to which result will be written
edata	ExpData instance containing observable data

Definition at line 1134 of file model.cpp.

Here is the call graph for this function:

9.10.3.87 fchi2()

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

chi-squared function

Parameters

it	time index
rdata	ReturnData instance to which result will be written

Definition at line 1149 of file model.cpp.

9.10.3.88 fsres()

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

residual sensitivity function

Parameters

it	time index
rdata	ReturnData instance to which result will be written
edata	ExpData instance containing observable data

Definition at line 1159 of file model.cpp.

Here is the call graph for this function:

9.10.3.89 fFIM()

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

fisher information matrix function

Parameters

it	time index
rdata	ReturnData instance to which result will be written

Definition at line 1175 of file model.cpp.

Here is the call graph for this function:

9.10.3.90 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 1191 of file model.cpp.

9.10.3.91 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 1197 of file model.cpp.

9.10.3.92 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 1248 of file model.cpp.

9.10.3.93 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 1302 of file model.cpp.

Here is the caller graph for this function:

9.10.3.94 hasParameterNames()

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

Returns

Definition at line 539 of file model.h.

Here is the call graph for this function:

9.10.3.95 getParameterNames()

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

Returns

the names

Reimplemented in Model_model_steadystate_scaled.

Definition at line 545 of file model.h.

Here is the caller graph for this function:

9.10.3.96 hasStateNames()

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

Returns

Definition at line 551 of file model.h.

Here is the call graph for this function:

9.10.3.97 getStateNames()

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

Returns

the names

Reimplemented in Model_model_steadystate_scaled.

Definition at line 557 of file model.h.

Here is the caller graph for this function:

9.10.3.98 hasFixedParameterNames()

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

Returns

Definition at line 563 of file model.h.

Here is the call graph for this function:

9.10.3.99 getFixedParameterNames()

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

Returns

the names

Reimplemented in Model_model_steadystate_scaled.

Definition at line 569 of file model.h.

Here is the caller graph for this function:

9.10.3.100 hasObservableNames()

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

Returns

Definition at line 575 of file model.h.

Here is the call graph for this function:

9.10.3.101 getObservableNames()

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

Returns

the names

Reimplemented in Model_model_steadystate_scaled.

Definition at line 581 of file model.h.

Here is the caller graph for this function:

9.10.3.102 hasParameterIds()

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

Returns

Definition at line 587 of file model.h.

Here is the call graph for this function:

9.10.3.103 getParameterIds()

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

Returns

the ids

Definition at line 593 of file model.h.

Here is the caller graph for this function:

9.10.3.104 hasStateIds()

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

Returns

Definition at line 599 of file model.h.

Here is the call graph for this function:

9.10.3.105 getStateIds()

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

Returns

the ids

Definition at line 605 of file model.h.

Here is the caller graph for this function:

9.10.3.106 hasFixedParameterIds()

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

Returns

Definition at line 611 of file model.h.

Here is the call graph for this function:

9.10.3.107 getFixedParameterIds()

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

Returns

the ids

Definition at line 617 of file model.h.

Here is the caller graph for this function:

9.10.3.108 hasObservableIds()

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

Returns

Definition at line 623 of file model.h.

Here is the call graph for this function:

9.10.3.109 getObservableIds()

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

Returns

the ids

Definition at line 629 of file model.h.

Here is the caller graph for this function:

9.10.3.110 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 707 of file model.h.

9.10.3.111 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 719 of file model.h.

9.10.3.112 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 734 of file model.h.

9.10.3.113 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 746 of file model.h.

9.10.3.114 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 759 of file model.h.

9.10.3.115 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 771 of file model.h.

9.10.3.116 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 784 of file model.h.

9.10.3.117 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 799 of file model.h.

9.10.3.118 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 812 of file model.h.

9.10.3.119 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 827 of file model.h.

9.10.3.120 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 841 of file model.h.

9.10.3.121 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 854 of file model.h.

9.10.3.122 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 868 of file model.h.

9.10.3.123 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 881 of file model.h.

9.10.3.124 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 896 of file model.h.

9.10.3.125 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 916 of file model.h.

9.10.3.126 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 934 of file model.h.

9.10.3.127 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 952 of file model.h.

9.10.3.128 fsigmay() [2/2]

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

model specific implementation of fsigmay

Parameters

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

Definition at line 963 of file model.h.

9.10.3.129 fdsigmaydp() [2/2]

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

model specific implementation of fsigmay

Parameters

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

Definition at line 974 of file model.h.

9.10.3.130 fsigmaz() [2/2]

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

model specific implementation of fsigmaz

Parameters

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

Definition at line 984 of file model.h.

9.10.3.131 fdsigmazdp() [2/2]

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

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

model specific implementation of fsigmaz

Parameters

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

Definition at line 995 of file model.h.

9.10.3.132 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 1008 of file model.h.

9.10.3.133 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 1020 of file model.h.

9.10.3.134 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 1032 of file model.h.

9.10.3.135 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 1045 of file model.h.

9.10.3.136 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 1060 of file model.h.

9.10.3.137 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 1074 of file model.h.

9.10.3.138 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 1089 of file model.h.

9.10.3.139 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 1102 of file model.h.

9.10.3.140 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 1116 of file model.h.

9.10.3.141 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 1129 of file model.h.

9.10.3.142 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 1141 of file model.h.

9.10.3.143 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 1153 of file model.h.

9.10.3.144 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 1207 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.145 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 1256 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.146 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 1238 of file model.cpp.

Here is the caller graph for this function:

9.10.3.147 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 1220 of file model.cpp.

Here is the caller graph for this function:

9.10.3.148 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 1229 of file model.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.10.3.149 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 1269 of file model.cpp.

Here is the caller graph for this function:

9.10.3.150 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 1278 of file model.cpp.

Here is the caller graph for this function:

9.10.3.151 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 1288 of file model.cpp.

Here is the call graph for this function:

9.10.3.152 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 1298 of file model.cpp.

Here is the call graph for this function:

9.10.4 Friends And Related Function Documentation

9.10.4.1 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

9.10.4.2 operator==

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

Parameters

a	
b	

Returns

Definition at line 1348 of file model.cpp.

9.10.5 Member Data Documentation

9.10.5.1 nx

```
const int nx
```

number of states

Definition at line 632 of file model.h.

9.10.5.2 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 634 of file model.h.

9.10.5.3 ny

```
const int ny
```

number of observables

Definition at line 636 of file model.h.

9.10.5.4 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 638 of file model.h.

9.10.5.5 nz

```
const int nz
```

number of event outputs

Definition at line 640 of file model.h.

9.10.5.6 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 642 of file model.h.

9.10.5.7 ne

```
const int ne
```

number of events

Definition at line 644 of file model.h.

9.10.5.8 nw

```
const int nw
```

number of common expressions

Definition at line 646 of file model.h.

9.10.5.9 ndwdx

```
const int ndwdx
```

number of derivatives of common expressions wrt x

Definition at line 648 of file model.h.

9.10.5.10 ndwdp

```
const int ndwdp
```

number of derivatives of common expressions wrt p

Definition at line 650 of file model.h.

9.10.5.11 nnz

```
const int nnz
```

number of nonzero entries in jacobian

Definition at line 652 of file model.h.

9.10.5.12 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 654 of file model.h.

9.10.5.13 ubw

```
const int ubw
```

upper bandwith of the jacobian

Definition at line 656 of file model.h.

9.10.5.14 lbw

```
const int lbw
```

lower bandwith of the jacobian

Definition at line 658 of file model.h.

9.10.5.15 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 661 of file model.h.

9.10.5.16 z2event

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

index indicating to which event an event output belongs

Definition at line 663 of file model.h.

9.10.5.17 idlist

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

flag array for DAE equations

Definition at line 665 of file model.h.

9.10.5.18 sigmay

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

data standard deviation for current timepoint (dimension: ny)

Definition at line 668 of file model.h.

9.10.5.19 dsigmaydp

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

parameter derivative of data standard deviation for current timepoint (dimension: nplist x ny, row-major)

Definition at line 670 of file model.h.

9.10.5.20 sigmaz

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

event standard deviation for current timepoint (dimension: nz)

Definition at line 672 of file model.h.

9.10.5.21 dsigmazdp

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

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

Definition at line 674 of file model.h.

9.10.5.22 dJydp

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

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

Definition at line 676 of file model.h.

9.10.5.23 dJzdp

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

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

Definition at line 678 of file model.h.

9.10.5.24 deltax

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

change in x at current timepoint (dimension: nx)

Definition at line 681 of file model.h.

9.10.5.25 deltasx

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

change in sx at current timepoint (dimension: nplist x nx, row-major)

Definition at line 683 of file model.h.

9.10.5.26 deltaxB

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

change in xB at current timepoint (dimension: nJ x nxtrue, row-major)

Definition at line 685 of file model.h.

9.10.5.27 deltaqB

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

change in qB at current timepoint (dimension: nJ x nplist, row-major)

Definition at line 687 of file model.h.

9.10.5.28 dxdotdp

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

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

Definition at line 690 of file model.h.

9.10.5.29 J

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

Sparse Jacobian (dimension: nnz)

Definition at line 1177 of file model.h.

9.10.5.30 my

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

current observable (dimension: nytrue)

Definition at line 1180 of file model.h.

9.10.5.31 mz

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

current event measurement (dimension: nztrue)

Definition at line 1182 of file model.h.

9.10.5.32 dJydy

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

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

Definition at line 1184 of file model.h.

9.10.5.33 dJydsigma

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

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

Definition at line 1186 of file model.h.

9.10.5.34 dJzdz

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

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

Definition at line 1189 of file model.h.

9.10.5.35 dJzdsigma

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

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

Definition at line 1191 of file model.h.

9.10.5.36 dJrzdz

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

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

Definition at line 1193 of file model.h.

9.10.5.37 dJrzdsigma

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

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

Definition at line 1195 of file model.h.

9.10.5.38 dzdx

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

state derivative of event output (dimension: nz $*$ nx, ordering = ?)

Definition at line 1197 of file model.h.

9.10.5.39 dzdp

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

parameter derivative of event output (dimension: nz ∗ nplist, ordering = ?)

Definition at line 1199 of file model.h.

9.10.5.40 drzdx

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

state derivative of event timepoint (dimension: nz ∗ nx, ordering = ?)

Definition at line 1201 of file model.h.

9.10.5.41 drzdp

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

parameter derivative of event timepoint (dimension: nz ∗ nplist, ordering = ?)

Definition at line 1203 of file model.h.

9.10.5.42 dydp

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

parameter derivative of observable (dimension: nplist ∗ ny, row-major)

Definition at line 1205 of file model.h.

9.10.5.43 dydx

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

state derivative of observable (dimension: ny ∗ nx, ordering = ?)

Definition at line 1208 of file model.h.

9.10.5.44 w

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

tempory storage of w data across functions (dimension: nw)

Definition at line 1210 of file model.h.

9.10.5.45 dwdx

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

tempory storage of sparse dwdx data across functions (dimension: ndwdx)

Definition at line 1212 of file model.h.

9.10.5.46 dwdp

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

tempory storage of sparse dwdp data across functions (dimension: ndwdp)

Definition at line 1214 of file model.h.

9.10.5.47 M

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

tempory storage of M data across functions (dimension: nx)

Definition at line 1216 of file model.h.

9.10.5.48 stau

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

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

Definition at line 1218 of file model.h.

9.10.5.49 h

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

flag indicating whether a certain heaviside function should be active or not (dimension: ne)

Definition at line 1222 of file model.h.

9.10.5.50 unscaledParameters

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

unscaled parameters (dimension: np)

Definition at line 1225 of file model.h.

9.10.5.51 originalParameters

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

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

Definition at line 1228 of file model.h.

9.10.5.52 fixedParameters

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

constants (dimension: nk)

Definition at line 1231 of file model.h.

9.10.5.53 plist_

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

indexes of parameters wrt to which sensitivities are computed (dimension nplist)

Definition at line 1234 of file model.h.

9.10.5.54 x0data

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

state initialisation (size nx)

Definition at line 1237 of file model.h.

9.10.5.55 sx0data

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

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

Definition at line 1240 of file model.h.

9.10.5.56 ts

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

timepoints (size nt)

Definition at line 1243 of file model.h.

9.10.5.57 nmaxevent

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

maximal number of events to track

Definition at line 1246 of file model.h.

9.10.5.58 pscale

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

parameter transformation of `originalParameters` (dimension np)

Definition at line 1249 of file model.h.

9.10.5.59 tstart

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

starting time

Definition at line 1252 of file model.h.

9.11 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

9.11.1 Detailed Description

Definition at line 24 of file model_dae.h.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 Model_DAE() [1/2]

```
Model_DAE ( )
```

default constructor

Definition at line 27 of file model_dae.h.

9.11.2.2 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.

9.11.3 Member Function Documentation

9.11.3.1 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:

9.11.3.2 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:

9.11.3.3 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:

9.11.3.4 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:

9.11.3.5 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:

9.11.3.6 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:

9.11.3.7 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:

9.11.3.8 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:

9.11.3.9 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:

9.11.3.10 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:

9.11.3.11 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:

9.11.3.12 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:

9.11.3.13 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:

9.11.3.14 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:

9.11.3.15 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:

9.11.3.16 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:

9.11.3.17 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:

9.11.3.18 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:

9.11.3.19 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:

9.11.3.20 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:

9.11.3.21 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.

9.11.3.22 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

9.11.3.23 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.

9.11.3.24 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

9.11.3.25 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.

9.11.3.26 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.

9.11.3.27 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.

9.11.3.28 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.

9.11.3.29 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.

9.11.3.30 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

9.11.3.31 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.

9.11.3.32 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.

9.11.3.33 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.

9.11.3.34 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.

9.11.3.35 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.

9.12 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

9.12.1 Detailed Description

Definition at line 23 of file model_ode.h.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 Model_ODE() [1/3]

```
Model_ODE ( )
```

default constructor

Definition at line 26 of file model_ode.h.

9.12.2.2 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.

9.12.2.3 Model_ODE() [3/3]

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

Parameters

other	

Definition at line 66 of file model_ode.h.

9.12.3 Member Function Documentation

9.12.3.1 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:

9.12.3.2 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:

9.12.3.3 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:

9.12.3.4 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:

9.12.3.5 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:

9.12.3.6 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:

9.12.3.7 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:

9.12.3.8 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:

9.12.3.9 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:

9.12.3.10 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:

9.12.3.11 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:

9.12.3.12 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:

9.12.3.13 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:

9.12.3.14 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:

9.12.3.15 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:

9.12.3.16 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:

9.12.3.17 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:

9.12.3.18 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:

9.12.3.19 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:

9.12.3.20 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:

9.12.3.21 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.

9.12.3.22 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

9.12.3.23 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.

9.12.3.24 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

9.12.3.25 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.

9.12.3.26 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.

9.12.3.27 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.

9.12.3.28 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.

9.12.3.29 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.

9.12.3.30 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

9.12.3.31 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.

9.12.3.32 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.

9.12.3.33 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.

9.12.3.34 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.

9.13 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)

9.13.1 Detailed Description

Definition at line 179 of file exception.h.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 NewtonFailure()

```
NewtonFailure (
            const char * msg )
```

constructor, simply calls AmiException constructor

Parameters

in	*msg*	

Definition at line 184 of file exception.h.

9.14 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 std::unique_ptr< 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

9.14.1 Detailed Description

Definition at line 24 of file newton_solver.h.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 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.

9.14.3 Member Function Documentation

9.14.3.1 getSolver()

```
std::unique_ptr< 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:

9.14.3.2 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:

9.14.3.3 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:

9.14.3.4 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:

9.14.3.5 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:

9.14.4 Member Data Documentation

9.14.4.1 maxlinsteps

```
int maxlinsteps = 0
```

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

Definition at line 58 of file newton_solver.h.

9.14.4.2 maxsteps

```
int maxsteps = 0
```

maximum number of allowed Newton steps for steady state computation

Definition at line 60 of file newton_solver.h.

9.14.4.3 atol

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

absolute tolerance

Definition at line 62 of file newton_solver.h.

9.14.4.4 rtol

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

relative tolerance

Definition at line 64 of file newton_solver.h.

9.14.4.5 t

```
realtype* t  [protected]
```

time variable

Definition at line 68 of file newton_solver.h.

9.14.4.6 model

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

pointer to the AMICI model object

Definition at line 70 of file newton_solver.h.

9.14.4.7 rdata

```
ReturnData* rdata [protected]
```

pointer to the return data object

Definition at line 72 of file newton_solver.h.

9.14.4.8 xdot

```
AmiVector xdot [protected]
```

right hand side AmiVector

Definition at line 74 of file newton_solver.h.

9.14.4.9 x

```
AmiVector* x [protected]
```

current state

Definition at line 76 of file newton_solver.h.

9.14.4.10 dx

```
AmiVector dx [protected]
```

current state time derivative (DAE)

Definition at line 78 of file newton_solver.h.

9.15 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

9.15.1 Detailed Description

Definition at line 87 of file newton_solver.h.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 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.

9.15.3 Member Function Documentation

9.15.3.1 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:

9.15.3.2 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:

9.16 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

9.16.1 Detailed Description

Definition at line 135 of file newton_solver.h.

9.16.2 Constructor & Destructor Documentation

9.16.2.1 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 322 of file newton_solver.cpp.

9.16.3 Member Function Documentation

9.16.3.1 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 360 of file newton_solver.cpp.

Here is the call graph for this function:

9.16.3.2 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 339 of file newton_solver.cpp.

9.16.3.3 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 373 of file newton_solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.17 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

9.17.1 Detailed Description

Definition at line 108 of file newton_solver.h.

9.17.2 Constructor & Destructor Documentation

9.17.2.1 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.

9.17.3 Member Function Documentation

9.17.3.1 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 288 of file newton_solver.cpp.

Here is the call graph for this function:

9.17.3.2 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:

9.18 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< realtype > FIM
- 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)

9.18.1 Detailed Description

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

Definition at line 28 of file rdata.h.

9.18.2 Constructor & Destructor Documentation

9.18.2.1 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:

9.18.3 Member Function Documentation

9.18.3.1 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 103 of file rdata.cpp.

Here is the call graph for this function:

9.18.3.2 invalidateLLH()

```
void invalidateLLH ( )
```

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

Definition at line 142 of file rdata.cpp.

Here is the caller graph for this function:

9.18.3.3 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 154 of file rdata.cpp.

Here is the call graph for this function:

9.18.4 Friends And Related Function Documentation

9.18.4.1 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

9.18.5 Member Data Documentation

9.18.5.1 ts

```
std::vector<realtype> ts
```

timepoints (dimension: nt)

Definition at line 48 of file rdata.h.

9.18.5.2 xdot

```
std::vector<realtype> xdot
```

time derivative (dimension: nx)

Definition at line 51 of file rdata.h.

9.18.5.3 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.

9.18.5.4 z

```
std::vector<realtype> z
```

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

Definition at line 58 of file rdata.h.

9.18.5.5 sigmaz

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

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

Definition at line 62 of file rdata.h.

9.18.5.6 sz

```
std::vector<realtype> sz
```

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

Definition at line 66 of file rdata.h.

9.18.5.7 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.

9.18.5.8 rz

```
std::vector<realtype> rz
```

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

Definition at line 73 of file rdata.h.

9.18.5.9 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.

9.18.5.10 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.

9.18.5.11 x

```
std::vector<realtype> x
```

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

Definition at line 84 of file rdata.h.

9.18.5.12 sx

```
std::vector<realtype> sx
```

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

Definition at line 88 of file rdata.h.

9.18.5.13 y

```
std::vector<realtype> y
```

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

Definition at line 91 of file rdata.h.

9.18.5.14 sigmay

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

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

Definition at line 94 of file rdata.h.

9.18.5.15 sy

```
std::vector<realtype> sy
```

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

Definition at line 98 of file rdata.h.

9.18.5.16 ssigmay

```
std::vector<realtype> ssigmay
```

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

Definition at line 102 of file rdata.h.

9.18.5.17 res

```
std::vector<realtype> res
```

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

Definition at line 105 of file rdata.h.

9.18.5.18 sres

```
std::vector<realtype> sres
```

parameter derivative of residual (dimension: nt∗ny x nplist, row-major)

Definition at line 109 of file rdata.h.

9.18.5.19 FIM

```
std::vector<realtype> FIM
```

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

Definition at line 113 of file rdata.h.

9.18.5.20 numsteps

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

number of integration steps forward problem (dimension: nt)

Definition at line 116 of file rdata.h.

9.18.5.21 numstepsB

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

number of integration steps backward problem (dimension: nt)

Definition at line 119 of file rdata.h.

9.18.5.22 numrhsevals

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

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

Definition at line 122 of file rdata.h.

9.18.5.23 numrhsevalsB

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

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

Definition at line 125 of file rdata.h.

9.18.5.24 numerrtestfails

```
std::vector<int> numerrtestfails
```

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

Definition at line 128 of file rdata.h.

9.18.5.25 numerrtestfailsB

```
std::vector<int> numerrtestfailsB
```

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

Definition at line 131 of file rdata.h.

9.18.5.26 numnonlinsolvconvfails

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

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

Definition at line 135 of file rdata.h.

9.18.5.27 numnonlinsolvconvfailsB

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

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

Definition at line 139 of file rdata.h.

9.18.5.28 order

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

employed order forward problem (dimension: nt)

Definition at line 142 of file rdata.h.

9.18.5.29 newton_status

```
int newton_status = 0
```

flag indicating success of Newton solver

Definition at line 145 of file rdata.h.

9.18.5.30 newton_time

```
double newton_time = 0.0
```

computation time of the Newton solver [s]

Definition at line 148 of file rdata.h.

9.18.5.31 newton_numsteps

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

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

Definition at line 151 of file rdata.h.

9.18.5.32 newton_numlinsteps

```
std::vector<int> newton_numlinsteps
```

number of linear steps by Newton step for steady state problem (length = newton_maxsteps $*$ 2)

Definition at line 154 of file rdata.h.

9.18.5.33 x0

```
std::vector<realtype> x0
```

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

Definition at line 157 of file rdata.h.

9.18.5.34 sx0

```
std::vector<realtype> sx0
```

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

Definition at line 160 of file rdata.h.

9.18.5.35 llh

```
realtype llh = 0.0
```

loglikelihood value

Definition at line 163 of file rdata.h.

9.18.5.36 chi2

```
realtype chi2 = 0.0
```

chi2 value

Definition at line 166 of file rdata.h.

9.18.5.37 sllh

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

parameter derivative of loglikelihood (dimension: nplist)

Definition at line 169 of file rdata.h.

9.18.5.38 s2llh

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

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

Definition at line 173 of file rdata.h.

9.18.5.39 status

```
int status = 0
```

status code

Definition at line 176 of file rdata.h.

9.18.5.40 np

```
const int np
```

total number of model parameters

Definition at line 180 of file rdata.h.

9.18.5.41 nk

```
const int nk
```

number of fixed parameters

Definition at line 182 of file rdata.h.

9.18.5.42 nx

```
const int nx
```

number of states

Definition at line 184 of file rdata.h.

9.18.5.43 nxtrue

```
const int nxtrue
```

number of states in the unaugmented system

Definition at line 186 of file rdata.h.

9.18.5.44 ny

```
const int ny
```

number of observables

Definition at line 188 of file rdata.h.

9.18.5.45 nytrue

```
const int nytrue
```

number of observables in the unaugmented system

Definition at line 190 of file rdata.h.

9.18.5.46 nz

```
const int nz
```

number of event outputs

Definition at line 192 of file rdata.h.

9.18.5.47 nztrue

```
const int nztrue
```

number of event outputs in the unaugmented system

Definition at line 194 of file rdata.h.

9.18.5.48 ne

```
const int ne
```

number of events

Definition at line 196 of file rdata.h.

9.18.5.49 nJ

```
const int nJ
```

dimension of the augmented objective function for 2nd order ASA

Definition at line 198 of file rdata.h.

9.18.5.50 nplist

```
const int nplist
```

number of parameter for which sensitivities were requested

Definition at line 201 of file rdata.h.

9.18.5.51 nmaxevent

```
const int nmaxevent
```

maximal number of occuring events (for every event type)

Definition at line 203 of file rdata.h.

9.18.5.52 nt

```
const int nt
```

number of considered timepoints

Definition at line 205 of file rdata.h.

9.18.5.53 newton_maxsteps

```
const int newton_maxsteps
```

maximal number of newton iterations for steady state calculation

Definition at line 207 of file rdata.h.

9.18.5.54 pscale

```
std::vector<AMICI_parameter_scaling> pscale
```

scaling of parameterization (lin,log,log10)

Definition at line 209 of file rdata.h.

9.18.5.55 o2mode

```
const AMICI_o2mode o2mode
```

flag indicating whether second order sensitivities were requested

Definition at line 211 of file rdata.h.

9.18.5.56 sensi

```
const AMICI_sensi_order sensi
```

sensitivity order

Definition at line 213 of file rdata.h.

9.18.5.57 sensi_meth

```
const AMICI_sensi_meth sensi_meth
```

sensitivity method

Definition at line 215 of file rdata.h.

9.19 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)

9.19.1 Detailed Description

Definition at line 166 of file exception.h.

9.19.2 Constructor & Destructor Documentation

9.19.2.1 SetupFailure()

```
SetupFailure (
            const char * msg )
```

constructor, simply calls AmiException constructor

Parameters

in	*msg*	

Definition at line 171 of file exception.h.

9.20 Solver Class Reference

```
#include <solver.h>
```

Public Member Functions

- Solver (const Solver &other)

  Solver copy constructor.
- virtual Solver ∗ clone () const =0

  Clone this instance.
- void setup (ForwardProblem ∗fwd, Model ∗model)

  setupAMIs initialises the ami memory object
- void setupAMIB (BackwardProblem ∗bwd, Model ∗model)
- virtual void getSens (realtype ∗tret, AmiVectorArray ∗yySout)=0
- void getDiagnosis (const int it, ReturnData ∗rdata)
- void getDiagnosisB (const int it, ReturnData ∗rdata, const BackwardProblem ∗bwd)
- virtual void getRootInfo (int ∗rootsfound)=0
- virtual void reInit (realtype t0, AmiVector ∗yy0, AmiVector ∗yp0)=0
- virtual void sensReInit (int ism, AmiVectorArray ∗yS0, AmiVectorArray ∗ypS0)=0
- virtual void calcIC (realtype tout1, AmiVector ∗x, AmiVector ∗dx)=0
- virtual void calcICB (int which, realtype tout1, AmiVector ∗xB, AmiVector ∗dxB)=0
- virtual int solve (realtype tout, AmiVector ∗yret, AmiVector ∗ypret, realtype ∗tret, int itask)=0
- virtual int solveF (realtype tout, AmiVector ∗yret, AmiVector ∗ypret, realtype ∗tret, int itask, int ∗ncheckPtr)=0
- virtual void solveB (realtype tBout, int itaskB)=0
- virtual void setStopTime (realtype tstop)=0
- virtual void reInitB (int which, realtype tB0, AmiVector ∗yyB0, AmiVector ∗ypB0)=0
- virtual void getB (int which, realtype ∗tret, AmiVector ∗yy, AmiVector ∗yp)=0
- virtual void getQuadB (int which, realtype ∗tret, AmiVector ∗qB)=0
- virtual void quadReInitB (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 create (int lmm, int iter)=0
- virtual void setSStolerances (double rtol, double atol)=0
- virtual void setSensSStolerances (double rtol, double ∗atol)=0
- virtual void setSensErrCon (bool error_corr)=0
- virtual void setQuadErrConB (int which, bool flag)=0
- virtual void setErrHandlerFn ()=0
- virtual void setUserData (Model ∗model)=0
- virtual void setUserDataB (int which, Model ∗model)=0
- virtual void setMaxNumSteps (long int mxsteps)=0
- virtual void setMaxNumStepsB (int which, long int mxstepsB)=0
- virtual void setStabLimDet (int stldet)=0
- virtual void setStabLimDetB (int which, int stldet)=0
- virtual void setId (Model ∗model)=0
- virtual void setSuppressAlg (bool flag)=0
- virtual void setSensParams (realtype ∗p, realtype ∗pbar, int ∗plist)=0
- virtual void getDky (realtype t, int k, AmiVector ∗dky)=0
- virtual void adjInit (long int steps, int interp)=0
- virtual void createB (int lmm, int iter, int ∗which)=0
- virtual void setSStolerancesB (int which, realtype relTolB, realtype absTolB)=0
- virtual void quadSStolerancesB (int which, realtype reltolQB, realtype abstolQB)=0
- virtual void dense (int nx)=0
- virtual void denseB (int which, int nx)=0
- virtual void band (int nx, int ubw, int lbw)=0
- virtual void bandB (int which, int nx, int ubw, int lbw)=0
- virtual void diag ()=0
- virtual void diagB (int which)=0
- virtual void spgmr (int prectype, int maxl)=0
- virtual void spgmrB (int which, int prectype, int maxl)=0
- virtual void spbcg (int prectype, int maxl)=0
- virtual void spbcgB (int which, int prectype, int maxl)=0
- virtual void sptfqmr (int prectype, int maxl)=0
- virtual void sptfqmrB (int which, int prectype, int maxl)=0
- virtual void klu (int nx, int nnz, int sparsetype)=0
- virtual void kluSetOrdering (int ordering)=0
- virtual void kluSetOrderingB (int which, int ordering)=0
- virtual void kluB (int which, int nx, int nnz, int sparsetype)=0
- virtual void getNumSteps (void ∗ami_mem, long int ∗numsteps)=0
- virtual void getNumRhsEvals (void ∗ami_mem, long int ∗numrhsevals)=0
- virtual void getNumErrTestFails (void ∗ami_mem, long int ∗numerrtestfails)=0
- virtual void getNumNonlinSolvConvFails (void ∗ami_mem, long int ∗numnonlinsolvconvfails)=0
- virtual void getLastOrder (void ∗ami_mem, int ∗order)=0
- virtual void ∗ getAdjBmem (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

- std::unique_ptr< void, std::function< void(void ∗)> > solverMemory
- 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.

9.20.1 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.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 Solver()

```
Solver (
            const Solver & other )
```

Parameters

other	

Definition at line 46 of file solver.h.

9.20.3 Member Function Documentation

9.20.3.1 clone()

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

Returns

The clone

9.20.3.2 setup()

```
void setup (
            ForwardProblem * fwd,
            Model * model )
```

Parameters

fwd	pointer to forward problem
model	pointer to the model object

Definition at line 24 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.20.3.3 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 97 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.20.3.4 getSens()

```
virtual void getSens (
            realtype * tret,
            AmiVectorArray * yySout )  [pure virtual]
```

getSens 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

9.20.3.5 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 195 of file solver.cpp.

Here is the call graph for this function:

9.20.3.6 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 223 of file solver.cpp.

Here is the call graph for this function:

9.20.3.7 getRootInfo()

```
virtual void getRootInfo (
            int * rootsfound )  [pure virtual]
```

getRootInfo extracts information which event occured

Parameters

rootsfound	array with flags indicating whether the respective event occured

9.20.3.8 reInit()

```
virtual void reInit (
            realtype t0,
            AmiVector * yy0,
            AmiVector * yp0 )  [pure virtual]
```

ReInit reinitializes the states in the solver after an event occurence

Parameters

t0	new timepoint
yy0	new state variables
yp0	new derivative state variables (DAE only)

Here is the caller graph for this function:

9.20.3.9 sensReInit()

```
virtual void sensReInit (
            int ism,
            AmiVectorArray * yS0,
            AmiVectorArray * ypS0 )  [pure virtual]
```

SensReInit 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)

Here is the caller graph for this function:

9.20.3.10 calcIC()

```
virtual void calcIC (
            realtype tout1,
```

```
AmiVector * x,
AmiVector * dx )  [pure virtual]
```

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

Parameters

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:

9.20.3.11 calcICB()

```
virtual void calcICB (
            int which,
            realtype tout1,
            AmiVector * xB,
            AmiVector * dxB )  [pure virtual]
```

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

Parameters

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:

9.20.3.12 solve()

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

Solve solves the forward problem until a predefined timepoint

Parameters

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:

9.20.3.13 solveF()

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

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

Parameters

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:

9.20.3.14 solveB()

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

SolveB 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:

9.20.3.15 setStopTime()

```
virtual void setStopTime (
            realtype tstop )  [pure virtual]
```

SetStopTime 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:

9.20.3.16 reInitB()

```
virtual void reInitB (
            int which,
            realtype tB0,
            AmiVector * yyB0,
            AmiVector * ypB0 )  [pure virtual]
```

ReInitB reinitializes the adjoint states after an event occurence

Parameters

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:

9.20.3.17 getB()

```
virtual void getB (
            int which,
            realtype * tret,
            AmiVector * yy,
            AmiVector * yp )  [pure virtual]
```

getB 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:

9.20.3.18 getQuadB()

```
virtual void getQuadB (
            int which,
            realtype * tret,
            AmiVector * qB )  [pure virtual]
```

getQuadB 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:

9.20.3.19 quadReInitB()

```
virtual void quadReInitB (
            int which,
            AmiVector * yQB0 )  [pure virtual]
```

ReInitB 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:

9.20.3.20 turnOffRootFinding()

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

turnOffRootFinding disables rootfinding Here is the caller graph for this function:

9.20.3.21 getSensitivityMethod()

```
AMICI_sensi_meth getSensitivityMethod ( ) const
```

sensitivity method

Returns

method enum

Definition at line 243 of file solver.h.

9.20.3.22 setSensitivityMethod()

```
void setSensitivityMethod (
            AMICI_sensi_meth sensi_meth )
```

Parameters

sensi_meth	

Definition at line 251 of file solver.h.

9.20.3.23 getNewtonMaxSteps()

```
int getNewtonMaxSteps ( ) const
```

Returns

Definition at line 259 of file solver.h.

Here is the caller graph for this function:

9.20.3.24 setNewtonMaxSteps()

```
void setNewtonMaxSteps (
            int newton_maxsteps )
```

Parameters

newton_maxsteps	

Definition at line 267 of file solver.h.

9.20.3.25 getNewtonPreequilibration()

```
bool getNewtonPreequilibration ( ) const
```

Returns

Definition at line 275 of file solver.h.

Here is the caller graph for this function:

9.20.3.26 setNewtonPreequilibration()

```
void setNewtonPreequilibration (
            bool newton_preeq )
```

Parameters

newton_preeq	

Definition at line 283 of file solver.h.

9.20.3.27 getNewtonMaxLinearSteps()

```
int getNewtonMaxLinearSteps ( ) const
```

Returns

Definition at line 292 of file solver.h.

Here is the caller graph for this function:

9.20.3.28 setNewtonMaxLinearSteps()

```
void setNewtonMaxLinearSteps (
            int newton_maxlinsteps )
```

Parameters

newton_maxlinsteps	

Definition at line 300 of file solver.h.

9.20.3.29 getSensitivityOrder()

```
AMICI_sensi_order getSensitivityOrder ( ) const
```

Returns

Definition at line 308 of file solver.h.

9.20.3.30 setSensitivityOrder()

```
void setSensitivityOrder (
            AMICI_sensi_order sensi )
```

Parameters

sensi	

Definition at line 316 of file solver.h.

9.20.3.31 getRelativeTolerance()

```
double getRelativeTolerance ( ) const
```

Returns

Definition at line 333 of file solver.h.

Here is the caller graph for this function:

9.20.3.32 setRelativeTolerance()

```
void setRelativeTolerance (
            double rtol )
```

Parameters

rtol	

Definition at line 341 of file solver.h.

9.20.3.33 getAbsoluteTolerance()

```
double getAbsoluteTolerance ( ) const
```

Returns

Definition at line 349 of file solver.h.

Here is the caller graph for this function:

9.20.3.34 setAbsoluteTolerance()

```
void setAbsoluteTolerance (
            double atol )
```

Parameters

atol	

Definition at line 357 of file solver.h.

9.20.3.35 getRelativeToleranceQuadratures()

```
double getRelativeToleranceQuadratures ( ) const
```

Returns

Definition at line 365 of file solver.h.

9.20.3.36 setRelativeToleranceQuadratures()

```
void setRelativeToleranceQuadratures (
            double rtol )
```

Parameters

rtol	

Definition at line 373 of file solver.h.

9.20.3.37 getAbsoluteToleranceQuadratures()

```
double getAbsoluteToleranceQuadratures ( ) const
```

Returns

Definition at line 381 of file solver.h.

9.20.3.38 setAbsoluteToleranceQuadratures()

```
void setAbsoluteToleranceQuadratures (
            double atol )
```

Parameters

atol	

Definition at line 389 of file solver.h.

9.20.3.39 getMaxSteps()

```
int getMaxSteps ( ) const
```

Returns

Definition at line 397 of file solver.h.

Here is the caller graph for this function:

9.20.3.40 setMaxSteps()

```
void setMaxSteps (
            int maxsteps )
```

Parameters

maxsteps	

Definition at line 405 of file solver.h.

9.20.3.41 getMaxStepsBackwardProblem()

```
int getMaxStepsBackwardProblem ( ) const
```

Returns

Definition at line 413 of file solver.h.

9.20.3.42 setMaxStepsBackwardProblem()

```
void setMaxStepsBackwardProblem (
            int maxsteps )
```

Parameters

maxsteps	

Definition at line 421 of file solver.h.

9.20.3.43 getLinearMultistepMethod()

```
LinearMultistepMethod getLinearMultistepMethod ( ) const
```

Returns

Definition at line 429 of file solver.h.

Here is the caller graph for this function:

9.20.3.44 setLinearMultistepMethod()

```
void setLinearMultistepMethod (
            LinearMultistepMethod lmm )
```

Parameters

lmm	

Definition at line 437 of file solver.h.

9.20.3.45 getNonlinearSolverIteration()

```
NonlinearSolverIteration getNonlinearSolverIteration ( ) const
```

Returns

Definition at line 448 of file solver.h.

Here is the caller graph for this function:

9.20.3.46 setNonlinearSolverIteration()

```
void setNonlinearSolverIteration (
            NonlinearSolverIteration iter )
```

Parameters

iter	

Definition at line 456 of file solver.h.

9.20.3.47 getInterpolationType()

```
InterpolationType getInterpolationType ( ) const
```

Returns

Definition at line 467 of file solver.h.

9.20.3.48 setInterpolationType()

```
void setInterpolationType (
            InterpolationType interpType )
```

Parameters

interpType	

Definition at line 475 of file solver.h.

9.20.3.49 getStateOrdering()

```
StateOrdering getStateOrdering ( ) const
```

Returns

Definition at line 483 of file solver.h.

Here is the caller graph for this function:

9.20.3.50 setStateOrdering()

```
void setStateOrdering (
            StateOrdering ordering )
```

Parameters

ordering	

Definition at line 491 of file solver.h.

9.20.3.51 getStabilityLimitFlag()

```
int getStabilityLimitFlag ( ) const
```

Returns

Definition at line 499 of file solver.h.

Here is the caller graph for this function:

9.20.3.52 setStabilityLimitFlag()

```
void setStabilityLimitFlag (
            int stldet )
```

Parameters

stldet	

Definition at line 507 of file solver.h.

9.20.3.53 getLinearSolver()

```
LinearSolver getLinearSolver ( ) const
```

Returns

Definition at line 515 of file solver.h.

Here is the caller graph for this function:

9.20.3.54 setLinearSolver()

```
void setLinearSolver (
            LinearSolver linsol )
```

Parameters

linsol	

Definition at line 523 of file solver.h.

9.20.3.55 getInternalSensitivityMethod()

```
InternalSensitivityMethod getInternalSensitivityMethod ( ) const
```

Returns

Definition at line 531 of file solver.h.

Here is the caller graph for this function:

9.20.3.56 setInternalSensitivityMethod()

```
void setInternalSensitivityMethod (
            InternalSensitivityMethod ism )
```

Parameters

ism	

Definition at line 539 of file solver.h.

9.20.3.57 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:

9.20.3.58 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:

9.20.3.59 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:

9.20.3.60 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:

9.20.3.61 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:

9.20.3.62 setDenseJacFn()

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

SetDenseJacFn sets the dense Jacobian function Here is the caller graph for this function:

9.20.3.63 setSparseJacFn()

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

SetSparseJacFn sets the sparse Jacobian function Here is the caller graph for this function:

9.20.3.64 setBandJacFn()

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

SetBandJacFn sets the banded Jacobian function Here is the caller graph for this function:

9.20.3.65 setJacTimesVecFn()

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

SetJacTimesVecFn sets the Jacobian vector multiplication function Here is the caller graph for this function:

9.20.3.66 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:

9.20.3.67 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:

9.20.3.68 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:

9.20.3.69 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:

9.20.3.70 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 153 of file solver.cpp.

9.20.3.71 create()

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

Create 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:

9.20.3.72 setSStolerances()

```
virtual void setSStolerances (
            double rtol,
            double atol )  [protected], [pure virtual]
```

SStolerances sets scalar relative and absolute tolerances for the forward problem

Parameters

rtol	relative tolerances
atol	absolute tolerances

Here is the caller graph for this function:

9.20.3.73 setSensSStolerances()

```
virtual void setSensSStolerances (
            double rtol,
            double * atol )  [protected], [pure virtual]
```

SensSStolerances activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

rtol	relative tolerances
atol	array of absolute tolerances for every sensitivy variable

Here is the caller graph for this function:

9.20.3.74 setSensErrCon()

```
virtual void setSensErrCon (
            bool error_corr )  [protected], [pure virtual]
```

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

Parameters

error_corr	activation flag

Here is the caller graph for this function:

9.20.3.75 setQuadErrConB()

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

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

Parameters

which	identifier of the backwards problem
flag	activation flag

Here is the caller graph for this function:

9.20.3.76 setErrHandlerFn()

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

SetErrHandlerFn attaches the error handler function (errMsgIdAndTxt) to the solver Here is the caller graph for this function:

9.20.3.77 setUserData()

```
virtual void setUserData (
            Model * model )  [protected], [pure virtual]
```

SetUserData attaches the user data instance (here this is a Model) to the forward problem

Parameters

model	Model instance,

Here is the caller graph for this function:

9.20.3.78 setUserDataB()

```
virtual void setUserDataB (
            int which,
            Model * model )  [protected], [pure virtual]
```

SetUserDataB attaches the user data instance (here this is a Model) to the backward problem

Parameters

which	identifier of the backwards problem
model	Model instance,

Here is the caller graph for this function:

9.20.3.79 setMaxNumSteps()

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

SetMaxNumSteps specifies the maximum number of steps for the forward problem

Parameters

mxsteps	number of steps

Here is the caller graph for this function:

9.20.3.80 setMaxNumStepsB()

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

SetMaxNumStepsB specifies the maximum number of steps for the forward problem

Parameters

which	identifier of the backwards problem
mxstepsB	number of steps

Here is the caller graph for this function:

9.20.3.81 setStabLimDet()

```
virtual void setStabLimDet (
            int stldet )  [protected], [pure virtual]
```

SetStabLimDet 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:

9.20.3.82 setStabLimDetB()

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

SetStabLimDetB 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:

9.20.3.83 setId()

```
virtual void setId (
            Model * model )  [protected], [pure virtual]
```

SetId specify algebraic/differential components (DAE only)

Parameters

model	model specification

Here is the caller graph for this function:

9.20.3.84 setSuppressAlg()

```
virtual void setSuppressAlg (
            bool flag )  [protected], [pure virtual]
```

SetId deactivates error control for algebraic components (DAE only)

Parameters

flag	deactivation flag

Here is the caller graph for this function:

9.20.3.85 setSensParams()

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

SetSensParams specifies the scaling and indexes for sensitivity computation

Parameters

p	paramaters
pbar	parameter scaling constants
plist	parameter index list

Here is the caller graph for this function:

9.20.3.86 getDky()

```
virtual void getDky (
            realtype t,
            int k,
            AmiVector * dky )  [protected], [pure virtual]
```

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

Parameters

t	timepoint
k	derivative order
dky	interpolated solution

9.20.3.87 adjInit()

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

AdjInit 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:

9.20.3.88 createB()

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

CreateB 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:

9.20.3.89 setSStolerancesB()

```
virtual void setSStolerancesB (
            int which,
```

```
            realtype relTolB,
            realtype absTolB )  [protected], [pure virtual]
```

SStolerancesB sets relative and absolute tolerances for the backward problem

Parameters

which	identifier of the backwards problem
relTolB	relative tolerances
absTolB	absolute tolerances

Here is the caller graph for this function:

9.20.3.90 quadSStolerancesB()

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

SStolerancesB sets relative and absolute tolerances for the quadrature backward problem

Parameters

which	identifier of the backwards problem
reltolQB	relative tolerances
abstolQB	absolute tolerances

Here is the caller graph for this function:

9.20.3.91 dense()

```
virtual void dense (
            int nx )  [protected], [pure virtual]
```

Dense attaches a dense linear solver to the forward problem

Parameters

nx	number of state variables

Here is the caller graph for this function:

9.20.3.92 denseB()

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

DenseB 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:

9.20.3.93 band()

```
virtual void band (
            int nx,
```

```
            int ubw,
            int lbw )  [protected], [pure virtual]
```

Band attaches a banded linear solver to the forward problem

Parameters

nx	number of state variables
ubw	upper matrix bandwidth
lbw	lower matrix bandwidth

Here is the caller graph for this function:

9.20.3.94 bandB()

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

BandB attaches a banded linear solver to the backward problem

Parameters

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:

9.20.3.95 diag()

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

Diag attaches a diagonal linear solver to the forward problem Here is the caller graph for this function:

9.20.3.96 diagB()

```
virtual void diagB (
            int which )  [protected], [pure virtual]
```

DiagB attaches a diagonal linear solver to the backward problem

Parameters

which	identifier of the backwards problem

Here is the caller graph for this function:

9.20.3.97 spgmr()

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

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

Parameters

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:

9.20.3.98 spgmrB()

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

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

Parameters

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:

9.20.3.99 spbcg()

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

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

Parameters

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:

9.20.3.100 spbcgB()

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

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

Parameters

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:

9.20.3.101 sptfqmr()

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

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

Parameters

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:

9.20.3.102 sptfqmrB()

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

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

Parameters

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:

9.20.3.103 klu()

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

KLU attaches a sparse linear solver to the forward problem

Parameters

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:

9.20.3.104 kluSetOrdering()

```
virtual void kluSetOrdering (
            int ordering )  [protected], [pure virtual]
```

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

Parameters

ordering	ordering algorithm to reduce fill 0:AMD 1:COLAMD 2: natural ordering

Here is the caller graph for this function:

9.20.3.105 kluSetOrderingB()

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

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

Parameters

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:

9.20.3.106 kluB()

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

KLUB attaches a sparse linear solver to the forward problem

Parameters

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:

9.20.3.107 getNumSteps()

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

getNumSteps 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:

9.20.3.108 getNumRhsEvals()

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

getNumRhsEvals 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:

9.20.3.109 getNumErrTestFails()

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

getNumErrTestFails 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:

9.20.3.110 getNumNonlinSolvConvFails()

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

getNumNonlinSolvConvFails 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:

9.20.3.111 getLastOrder()

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

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:

9.20.3.112 getAdjBmem()

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

getAdjBmem 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:

9.20.3.113 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 248 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.20.3.114 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 325 of file solver.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.20.4 Friends And Related Function Documentation

9.20.4.1 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

9.20.4.2 operator==

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

Parameters

a	
b	

Returns

Definition at line 399 of file solver.cpp.

9.20.5 Member Data Documentation

9.20.5.1 solverMemory

```
std::unique_ptr<void, std::function<void(void *)> > solverMemory [protected]
```

pointer to solver memory block

Definition at line 1053 of file solver.h.

9.20.5.2 solverWasCalled

```
bool solverWasCalled = false [protected]
```

flag indicating whether the solver was called

Definition at line 1056 of file solver.h.

9.21 SteadystateProblem Class Reference

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

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

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, int newton_status, double run_time, int it)
- void getSteadystateSimulation (ReturnData *rdata, Solver *solver, Model *model, int it)
- std::unique_ptr< void, std::function< void(void *)> > createSteadystateSimSolver (Solver *solver, Model *model, realtype tstart)
- SteadystateProblem (realtype *t, AmiVector *x, AmiVectorArray *sx)

9.21.1 Detailed Description

Definition at line 24 of file steadystateproblem.h.

9.21.2 Constructor & Destructor Documentation

9.21.2.1 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 49 of file steadystateproblem.h.

9.21.3 Member Function Documentation

9.21.3.1 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 20 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.21.3.2 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:

9.21.3.3 getNewtonOutput()

```
void getNewtonOutput (
            ReturnData * rdata,
            int newton_status,
            double run_time,
            int it )
```

Stores output of workSteadyStateProblem in return data

Parameters

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 198 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.21.3.4 getSteadystateSimulation()

```
void getSteadystateSimulation (
            ReturnData * rdata,
            Solver * solver,
            Model * model,
            int it )
```

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

Parameters

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 233 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.21.3.5 createSteadystateSimSolver()

```
std::unique_ptr< void, std::function< void(void *)> > createSteadystateSimSolver (
            Solver * solver,
            Model * model,
            realtype tstart )
```

New CVode object for preequilibration simulation is created

Parameters

in	*solver*	pointer to the AMICI solver object **Type**: Solver
in	*model*	pointer to the AMICI model object **Type**: Model
in	*tstart*	time point for starting Newton simulation **Type**: realtype

Definition at line 304 of file steadystateproblem.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

9.22 Model_model_steadystate_scaled Class Reference

AMICI-generated model subclass.

```
#include <model_steadystate_scaled.h>
```

Inheritance diagram for Model_model_steadystate_scaled:

Public Member Functions

- Model_model_steadystate_scaled ()

  Default constructor.
- virtual amici::Model ∗ clone () const override

  Clone this model instance.
- virtual void fJ (realtype ∗J, const realtype t, const realtype ∗x, const double ∗p, const double ∗k, const realtype ∗h, const realtype ∗w, const realtype ∗dwdx) override
- 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) override
- 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) override
- 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) override
- 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) override
- virtual void fJrz (double ∗nllh, const int iz, const realtype ∗p, const realtype ∗k, const double ∗rz, const double ∗sigmaz) override
- 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) override
- 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) override
- virtual void fJy (double ∗nllh, const int iy, const realtype ∗p, const realtype ∗k, const double ∗y, const double ∗sigmay, const double ∗my) override
- virtual void fJz (double ∗nllh, const int iz, const realtype ∗p, const realtype ∗k, const double ∗z, const double ∗sigmaz, const double ∗mz) override
- virtual void fdJrzdsigma (double ∗dJrzdsigma, const int iz, const realtype ∗p, const realtype ∗k, const double ∗rz, const double ∗sigmaz) override
- virtual void fdJrzdz (double ∗dJrzdz, const int iz, const realtype ∗p, const realtype ∗k, const double ∗rz, const double ∗sigmaz) override
- virtual void fdJydsigma (double ∗dJydsigma, const int iy, const realtype ∗p, const realtype ∗k, const double ∗y, const double ∗sigmay, const double ∗my) override

- virtual void fdJydy (double *dJydy, const int iy, const realtype *p, const realtype *k, const double *y, const double *sigmay, const double *my) override
- virtual void fdJzdsigma (double *dJzdsigma, const int iz, const realtype *p, const realtype *k, const double *z, const double *sigmaz, const double *mz) override
- virtual void fdJzdz (double *dJzdz, const int iz, const realtype *p, const realtype *k, const double *z, const double *sigmaz, const double *mz) override
- virtual void fdeltaqB (double *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) override
- virtual void fdeltasx (double *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) override
- virtual void fdeltax (double *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) override
- virtual void fdeltaxB (double *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) override
- virtual void fdrzdp (double *drzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip) override
- virtual void fdrzdx (double *drzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h) override
- virtual void fdsigmaydp (double *dsigmaydp, const realtype t, const realtype *p, const realtype *k, const int ip) override
- virtual void fdsigmazdp (double *dsigmazdp, const realtype t, const realtype *p, const realtype *k, const int ip) override
- virtual void fdwdp (realtype *dwdp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w) override
- virtual void fdwdx (realtype *dwdx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const realtype *w) override
- 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) override
- virtual void fdydx (double *dydx, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h) override
- virtual void fdydp (double *dydp, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip) override
- virtual void fdzdp (double *dzdp, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h, const int ip) override
- virtual void fdzdx (double *dzdx, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h) override
- 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) override
- virtual void froot (realtype *root, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h) override
- virtual void frz (double *rz, const int ie, const realtype t, const realtype *x, const realtype *p, const realtype *k, const realtype *h) override
- virtual void fsigmay (double *sigmay, const realtype t, const realtype *p, const realtype *k) override
- virtual void fsigmaz (double *sigmaz, const realtype t, const realtype *p, const realtype *k) override
- virtual void fsrz (double *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) override
- virtual void fstau (double *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) override
- virtual void fsx0 (realtype *sx0, const realtype t, const realtype *x0, const realtype *p, const realtype *k, const int ip) override
- 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) override
- virtual void fsz (double *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) override

- virtual void fw (realtype ∗w, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h) override
- virtual void fx0 (realtype ∗x0, const realtype t, const realtype ∗p, const realtype ∗k) override
- 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) override
- virtual void fxdot (realtype ∗xdot, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h, const realtype ∗w) override
- virtual void fy (double ∗y, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h) override
- virtual void fz (double ∗z, const int ie, const realtype t, const realtype ∗x, const realtype ∗p, const realtype ∗k, const realtype ∗h) override
- virtual std::vector< std::string > getParameterNames () const

  Get names of the model parameters.
- virtual std::vector< std::string > getStateNames () const

  Get names of the model states.
- virtual std::vector< std::string > getFixedParameterNames () const

  Get names of the fixed model parameters.
- virtual std::vector< std::string > getObservableNames () const

  Get names of the observables.

Additional Inherited Members

9.22.1 Detailed Description

Definition at line 46 of file model_steadystate_scaled.h.

9.22.2 Member Function Documentation

9.22.2.1 clone()

```
virtual amici::Model* clone ( ) const  [override], [virtual]
```

Returns

A deep copy of this instance.

Implements Model.

Definition at line 80 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.2 fJ()

```
virtual void fJ (
            realtype * J,
            const realtype t,
            const realtype * x,
            const double * p,
            const double * k,
            const realtype * h,
            const realtype * w,
            const realtype * dwdx )  [override], [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
dwdx	derivative of w wrt x

Definition at line 92 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.3 fJB()

```
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 )  [override], [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 107 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.4 fJDiag()

```
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 )  [override], [virtual]
```

model specific implementation for fJDiag

Parameters

JDiag	Matrix to which the Jacobian will be written

Parameters

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 121 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.5 fJSparse()

```
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 )  [override], [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

Definition at line 135 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.6 fJSparseB()

```
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 )  [override], [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 150 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.7 fJrz()

```
virtual void fJrz (
            double * nllh,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * rz,
            const double * sigmaz )  [override], [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 162 of file model_steadystate_scaled.h.

9.22.2.8 fJv()

```
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 )  [override], [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 176 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.9 fJvB()

```
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 )  [override], [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

Parameters

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 192 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.10 fJy()

```
virtual void fJy (
            double * nllh,
            const int iy,
            const realtype * p,
            const realtype * k,
            const double * y,
            const double * sigmay,
            const double * my )  [override], [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 205 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.11 fJz()

```
virtual void fJz (
            double * nllh,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * z,
            const double * sigmaz,
            const double * mz )  [override], [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 218 of file model_steadystate_scaled.h.

9.22.2.12 fdJrzdsigma()

```
virtual void fdJrzdsigma (
            double * dJrzdsigma,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * rz,
            const double * sigmaz )  [override], [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 230 of file model_steadystate_scaled.h.

9.22.2.13 fdJrzdz()

```
virtual void fdJrzdz (
            double * dJrzdz,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * rz,
            const double * sigmaz )  [override], [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 241 of file model_steadystate_scaled.h.

9.22.2.14 fdJydsigma()

```
virtual void fdJydsigma (
            double * dJydsigma,
            const int iy,
            const realtype * p,
            const realtype * k,
            const double * y,
            const double * sigmay,
            const double * my )  [override], [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 254 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.15 fdJydy()

```
virtual void fdJydy (
            double * dJydy,
            const int iy,
            const realtype * p,
            const realtype * k,
            const double * y,
            const double * sigmay,
            const double * my )  [override], [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 267 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.16 fdJzdsigma()

```
virtual void fdJzdsigma (
            double * dJzdsigma,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * z,
            const double * sigmaz,
            const double * mz )  [override], [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 281 of file model_steadystate_scaled.h.

9.22.2.17 fdJzdz()

```
virtual void fdJzdz (
            double * dJzdz,
            const int iz,
            const realtype * p,
            const realtype * k,
            const double * z,
            const double * sigmaz,
            const double * mz )  [override], [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 293 of file model_steadystate_scaled.h.

9.22.2.18 fdeltaqB()

```
virtual void fdeltaqB (
            double * 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 )  [override], [virtual]
```

model specific implementation of fdeltasx

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 309 of file model_steadystate_scaled.h.

9.22.2.19 fdeltasx()

```
virtual void fdeltasx (
            double * 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 )  [override], [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 327 of file model_steadystate_scaled.h.

9.22.2.20 fdeltax()

```
virtual void fdeltax (
            double * 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 )  [override], [virtual]
```

model specific implementation of fdeltax

Parameters

deltax	state update
t	current time

Parameters

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 341 of file model_steadystate_scaled.h.

9.22.2.21 fdeltaxB()

```
virtual void fdeltaxB (
            double * 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 )  [override], [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 356 of file model_steadystate_scaled.h.

9.22.2.22 fdrzdp()

```
virtual void fdrzdp (
            double * drzdp,
            const int ie,
```

```
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [override], [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 369 of file model_steadystate_scaled.h.

9.22.2.23 fdrzdx()

```
virtual void fdrzdx (
            double * drzdx,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 381 of file model_steadystate_scaled.h.

9.22.2.24 fdsigmaydp()

```
virtual void fdsigmaydp (
            double * dsigmaydp,
```

```
            const realtype t,
            const realtype * p,
            const realtype * k,
            const int ip )  [override], [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 391 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.25 fdsigmazdp()

```
virtual void fdsigmazdp (
            double * dsigmazdp,
            const realtype t,
            const realtype * p,
            const realtype * k,
            const int ip )  [override], [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 403 of file model_steadystate_scaled.h.

9.22.2.26 fdwdp()

```
virtual void fdwdp (
            realtype * dwdp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const realtype * w )  [override], [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 415 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.27 fdwdx()

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

```
            const realtype * k,
            const realtype * h,
            const realtype * w )  [override], [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 428 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.28 fdxdotdp()

```
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 )  [override], [virtual]
```

model specific implementation of fdxdotdp

Parameters

dxdotdp	partial derivative xdot wrt p
t	timepoint

Parameters

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 443 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.29 fdydx()

```
virtual void fdydx (
            double * dydx,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 455 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.30 fdydp()

```
virtual void fdydp (
            double * dydp,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [override], [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 468 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.31 fdzdp()

```
virtual void fdzdp (
            double * dzdp,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h,
            const int ip )  [override], [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 482 of file model_steadystate_scaled.h.

9.22.2.32 fdzdx()

```
virtual void fdzdx (
            double * dzdx,
            const int ie,
            const realtype t,
            const realtype * x,
```

```
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 494 of file model_steadystate_scaled.h.

9.22.2.33 fqBdot()

```
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 )  [override], [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 509 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.34 froot()

```
virtual void froot (
            realtype * root,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 521 of file model_steadystate_scaled.h.

9.22.2.35 frz()

```
virtual void frz (
            double * rz,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 533 of file model_steadystate_scaled.h.

9.22.2.36 fsigmay()

```
virtual void fsigmay (
            double * sigmay,
            const realtype t,
            const realtype * p,
            const realtype * k )  [override], [virtual]
```

model specific implementation of fsigmay

Parameters

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

Definition at line 542 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.37 fsigmaz()

```
virtual void fsigmaz (
            double * sigmaz,
            const realtype t,
            const realtype * p,
            const realtype * k )  [override], [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 552 of file model_steadystate_scaled.h.

9.22.2.38 fsrz()

```
virtual void fsrz (
            double * 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 )  [override], [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 566 of file model_steadystate_scaled.h.

9.22.2.39 fstau()

```
virtual void fstau (
            double * 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 )  [override], [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 580 of file model_steadystate_scaled.h.

9.22.2.40 fsx0()

```
virtual void fsx0 (
            realtype * sx0,
            const realtype t,
            const realtype * x0,
            const realtype * p,
            const realtype * k,
            const int ip )  [override], [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 591 of file model_steadystate_scaled.h.

9.22.2.41 fsxdot()

```
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 )  [override], [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 608 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.42 fsz()

```
virtual void fsz (
            double * 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 )  [override], [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 623 of file model_steadystate_scaled.h.

9.22.2.43 fw()

```
virtual void fw (
            realtype * w,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 634 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.44 fx0()

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

model specific implementation of fx0

Parameters

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

Definition at line 644 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.45 fxBdot()

```
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 )  [override], [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 659 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.46 fxdot()

```
virtual void fxdot (
            realtype * xdot,
            const realtype t,
            const realtype * x,
            const realtype * p,
```

```
            const realtype * k,
            const realtype * h,
            const realtype * w )  [override], [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

Definition at line 672 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.47 fy()

```
virtual void fy (
            double * y,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 684 of file model_steadystate_scaled.h.

Here is the call graph for this function:

9.22.2.48 fz()

```
virtual void fz (
            double * z,
            const int ie,
            const realtype t,
            const realtype * x,
            const realtype * p,
            const realtype * k,
            const realtype * h )  [override], [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 697 of file model_steadystate_scaled.h.

9.22.2.49 getParameterNames()

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

Returns

the names

Reimplemented from Model.

Definition at line 706 of file model_steadystate_scaled.h.

9.22.2.50 getStateNames()

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

Returns

the names

Reimplemented from Model.

Definition at line 719 of file model_steadystate_scaled.h.

9.22.2.51 getFixedParameterNames()

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

Returns

the names

Reimplemented from Model.

Definition at line 727 of file model_steadystate_scaled.h.

9.22.2.52 getObservableNames()

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

Returns

the names

Reimplemented from Model.

Definition at line 733 of file model_steadystate_scaled.h.

9.23 PyHeapTypeObject Struct Reference

Public Attributes

- PyTypeObject **type**
- PyNumberMethods **as_number**
- PyMappingMethods **as_mapping**
- PySequenceMethods **as_sequence**
- PyBufferProcs **as_buffer**
- PyObject ∗ **name**
- PyObject ∗ **slots**

9.23.1 Detailed Description

Definition at line 955 of file model_steadystate_scaled_wrap.cpp.

9.24 less< PyObject ∗ > Struct Template Reference

Public Member Functions

- bool **operator()** (PyObject ∗v, PyObject ∗w) const

9.24.1 Detailed Description

template<>
struct std::less< PyObject ∗ >

Definition at line 3544 of file model_steadystate_scaled_wrap.cpp.

9.25 less< swig::SwigPtr_PyObject > Struct Template Reference

Public Member Functions

- bool **operator()** (const swig::SwigPtr_PyObject &v, const swig::SwigPtr_PyObject &w) const

9.25.1 Detailed Description

template<>
struct std::less< swig::SwigPtr_PyObject >

Definition at line 3569 of file model_steadystate_scaled_wrap.cpp.

9.26 less< swig::SwigVar_PyObject > Struct Template Reference

Public Member Functions

- bool **operator()** (const swig::SwigVar_PyObject &v, const swig::SwigVar_PyObject &w) const

9.26.1 Detailed Description

template<>
struct std::less< swig::SwigVar_PyObject >

Definition at line 3579 of file model_steadystate_scaled_wrap.cpp.

9.27 from_oper< ValueType > Struct Template Reference

Public Types

- typedef const ValueType & **argument_type**
- typedef PyObject * **result_type**

Public Member Functions

- result_type **operator()** (argument_type v) const

9.27.1 Detailed Description

template<class ValueType>
struct swig::from_oper< ValueType >

Definition at line 3938 of file model_steadystate_scaled_wrap.cpp.

9.28 noconst_traits< Type > Struct Template Reference

Public Types

- typedef Type **noconst_type**

9.28.1 Detailed Description

template<class Type>
struct swig::noconst_traits< Type >

Definition at line 3269 of file model_steadystate_scaled_wrap.cpp.

9.29 noconst_traits< const Type > Struct Template Reference

Public Types

- typedef Type **noconst_type**

9.29.1 Detailed Description

template<class Type>
struct swig::noconst_traits< const Type >

Definition at line 3274 of file model_steadystate_scaled_wrap.cpp.

9.30 pointer_category Struct Reference

9.30.1 Detailed Description

Definition at line 3281 of file model_steadystate_scaled_wrap.cpp.

9.31 stop_iteration Struct Reference

9.31.1 Detailed Description

Definition at line 3138 of file model_steadystate_scaled_wrap.cpp.

9.32 SwigPtr_PyObject Class Reference

Inheritance diagram for SwigPtr_PyObject:

Public Member Functions

- **SwigPtr_PyObject** (const SwigPtr_PyObject &item)
- **SwigPtr_PyObject** (PyObject *obj, bool initial_ref=true)
- SwigPtr_PyObject & **operator=** (const SwigPtr_PyObject &item)
- **operator PyObject** * () const
- PyObject * **operator->** () const

Protected Attributes

- PyObject * **_obj**

9.32.1 Detailed Description

Definition at line 3062 of file model_steadystate_scaled_wrap.cpp.

9.33 SwigPyIterator Struct Reference

Inheritance diagram for SwigPyIterator:

Public Member Functions

- virtual PyObject * **value** () const =0
- virtual SwigPyIterator * **incr** (size_t n=1)=0
- virtual SwigPyIterator * **decr** (size_t=1)
- virtual ptrdiff_t **distance** (const SwigPyIterator &) const
- virtual bool **equal** (const SwigPyIterator &) const
- virtual SwigPyIterator * **copy** () const =0
- PyObject * **next** ()
- PyObject * **__next__** ()
- PyObject * **previous** ()
- SwigPyIterator * **advance** (ptrdiff_t n)
- bool **operator==** (const SwigPyIterator &x) const
- bool **operator!=** (const SwigPyIterator &x) const
- SwigPyIterator & **operator+=** (ptrdiff_t n)
- SwigPyIterator & **operator-=** (ptrdiff_t n)
- SwigPyIterator * **operator+** (ptrdiff_t n) const
- SwigPyIterator * **operator-** (ptrdiff_t n) const
- ptrdiff_t **operator-** (const SwigPyIterator &x) const

Static Public Member Functions

- static swig_type_info ∗ **descriptor** ()

Protected Member Functions

- **SwigPyIterator** (PyObject ∗seq)

9.33.1 Detailed Description

Definition at line 3141 of file model_steadystate_scaled_wrap.cpp.

9.34 SwigPyIterator_T< OutIterator > Class Template Reference

Inheritance diagram for SwigPyIterator_T< OutIterator >:

Public Types

- typedef OutIterator **out_iterator**
- typedef std::iterator_traits< out_iterator >::value_type **value_type**
- typedef SwigPyIterator_T< out_iterator > **self_type**

Public Member Functions

- **SwigPyIterator_T** (out_iterator curr, PyObject ∗seq)
- const out_iterator & **get_current** () const
- bool **equal** (const SwigPyIterator &iter) const
- ptrdiff_t **distance** (const SwigPyIterator &iter) const

Protected Attributes

- out_iterator **current**

Additional Inherited Members

9.34.1 Detailed Description

template<typename OutIterator>
class swig::SwigPyIterator_T< OutIterator >

Definition at line 3895 of file model_steadystate_scaled_wrap.cpp.

9.35 SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper > Class Template Reference

Inheritance diagram for SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper >:

Public Types

- typedef OutIterator **out_iterator**
- typedef ValueType **value_type**
- typedef SwigPyIterator_T< out_iterator > **base**
- typedef SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper > **self_type**

Public Member Functions

- **SwigPyIteratorClosed_T** (out_iterator curr, out_iterator first, out_iterator last, PyObject *seq)
- PyObject * **value** () const
- SwigPyIterator * **copy** () const
- SwigPyIterator * **incr** (size_t n=1)
- SwigPyIterator * **decr** (size_t n=1)

Public Attributes

- FromOper **from**

Additional Inherited Members

9.35.1 Detailed Description

template<typename OutIterator, typename ValueType = typename std::iterator_traits<OutIterator>::value_type, typename FromOper = from_oper<ValueType>>
class swig::SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper >

Definition at line 3994 of file model_steadystate_scaled_wrap.cpp.

9.36 SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper > Class Template Reference

Inheritance diagram for SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper >:

Public Types

- typedef OutIterator **out_iterator**
- typedef ValueType **value_type**
- typedef SwigPyIterator_T< out_iterator > **base**
- typedef SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper > **self_type**

Public Member Functions

- **SwigPyIteratorOpen_T** (out_iterator curr, PyObject *seq)
- PyObject * **value** () const
- SwigPyIterator * **copy** () const
- SwigPyIterator * **incr** (size_t n=1)
- SwigPyIterator * **decr** (size_t n=1)

Public Attributes

- FromOper **from**

Additional Inherited Members

9.36.1 Detailed Description

template<typename OutIterator, typename ValueType = typename std::iterator_traits<OutIterator>::value_type, typename FromOper = from_oper<ValueType>>
class swig::SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper >

Definition at line 3951 of file model_steadystate_scaled_wrap.cpp.

9.37 SwigPySequence_ArrowProxy< T > Struct Template Reference

Public Member Functions

- **SwigPySequence_ArrowProxy** (const T &x)
- const T * **operator->** () const
- **operator const T** * () const

Public Attributes

- T **m_value**

9.37.1 Detailed Description

template<class T>
struct swig::SwigPySequence_ArrowProxy< T >

Definition at line 4106 of file model_steadystate_scaled_wrap.cpp.

9.38 SwigPySequence_Cont< T > Struct Template Reference

Public Types

- typedef SwigPySequence_Ref< T > **reference**
- typedef const SwigPySequence_Ref< T > **const_reference**
- typedef T **value_type**
- typedef T * **pointer**
- typedef Py_ssize_t **difference_type**
- typedef size_t **size_type**
- typedef const pointer **const_pointer**
- typedef SwigPySequence_InputIterator< T, reference > **iterator**
- typedef SwigPySequence_InputIterator< T, const_reference > **const_iterator**

Public Member Functions

- **SwigPySequence_Cont** (PyObject *seq)
- size_type **size** () const
- bool **empty** () const
- iterator **begin** ()
- const_iterator **begin** () const
- iterator **end** ()
- const_iterator **end** () const
- reference **operator[]** (difference_type n)
- const_reference **operator[]** (difference_type n) const
- bool **check** (bool set_err=true) const

9.38.1 Detailed Description

template<class T>
struct swig::SwigPySequence_Cont< T >

Definition at line 4211 of file model_steadystate_scaled_wrap.cpp.

9.39 SwigPySequence_InputIterator< T, Reference > Struct Template Reference

Public Types

- typedef SwigPySequence_InputIterator< T, Reference > **self**
- typedef std::random_access_iterator_tag **iterator_category**
- typedef Reference **reference**
- typedef T **value_type**
- typedef T * **pointer**
- typedef Py_ssize_t **difference_type**

Public Member Functions

- **SwigPySequence_InputIterator** (PyObject *seq, Py_ssize_t index)
- reference **operator*** () const
- SwigPySequence_ArrowProxy< T > **operator->** () const
- bool **operator==** (const self &ri) const
- bool **operator!=** (const self &ri) const
- self & **operator++** ()
- self & **operator--** ()
- self & **operator+=** (difference_type n)
- self **operator+** (difference_type n) const
- self & **operator-=** (difference_type n)
- self **operator-** (difference_type n) const
- difference_type **operator-** (const self &ri) const
- bool **operator<** (const self &ri) const
- reference **operator[]** (difference_type n) const

9.39.1 Detailed Description

template<class T, class Reference>
struct swig::SwigPySequence_InputIterator< T, Reference >

Definition at line 4115 of file model_steadystate_scaled_wrap.cpp.

9.40 SwigPySequence_Ref< T > Struct Template Reference

Public Member Functions

- **SwigPySequence_Ref** (PyObject *seq, Py_ssize_t index)
- **operator T** () const
- SwigPySequence_Ref & **operator=** (const T &v)

9.40.1 Detailed Description

template<class T>
struct swig::SwigPySequence_Ref< T >

Definition at line 4070 of file model_steadystate_scaled_wrap.cpp.

9.41 SwigVar_PyObject Struct Reference

Inheritance diagram for SwigVar_PyObject:

Public Member Functions

- **SwigVar_PyObject** (PyObject *obj=0)
- SwigVar_PyObject & **operator=** (PyObject *obj)

Additional Inherited Members

9.41.1 Detailed Description

Definition at line 3118 of file model_steadystate_scaled_wrap.cpp.

9.42 traits< Type > Struct Template Reference

9.42.1 Detailed Description

template<class Type>
struct swig::traits< Type >

Definition at line 3287 of file model_steadystate_scaled_wrap.cpp.

9.43 traits< PyObject * > Struct Template Reference

Public Types

- typedef value_category **category**

Static Public Member Functions

- static const char * **type_name** ()

9.43.1 Detailed Description

template<>
struct swig::traits< PyObject * >

Definition at line 3591 of file model_steadystate_scaled_wrap.cpp.

9.44 traits< Type * > Struct Template Reference

Public Types

- typedef pointer_category **category**

Static Public Member Functions

- static std::string **make_ptr_name** (const char *name)
- static const char * **type_name** ()

9.44.1 Detailed Description

template<class Type>
struct swig::traits< Type * >

Definition at line 3327 of file model_steadystate_scaled_wrap.cpp.

9.45 traits_as< Type, Category > Struct Template Reference

9.45.1 Detailed Description

template<class Type, class Category>
struct swig::traits_as< Type, Category >

Definition at line 3341 of file model_steadystate_scaled_wrap.cpp.

9.46 traits_as< Type *, pointer_category > Struct Template Reference

Static Public Member Functions

- static Type * **as** (PyObject *obj, bool throw_error)

9.46.1 Detailed Description

template<class Type>
struct swig::traits_as< Type *, pointer_category >

Definition at line 3496 of file model_steadystate_scaled_wrap.cpp.

9.47 traits_as< Type, pointer_category > Struct Template Reference

Static Public Member Functions

- static Type **as** (PyObject *obj, bool throw_error)

9.47.1 Detailed Description

template<class Type>
struct swig::traits_as< Type, pointer_category >

Definition at line 3470 of file model_steadystate_scaled_wrap.cpp.

9.48 traits_as< Type, value_category > Struct Template Reference

Static Public Member Functions

- static Type **as** (PyObject *obj, bool throw_error)

9.48.1 Detailed Description

template<class Type>
struct swig::traits_as< Type, value_category >

Definition at line 3455 of file model_steadystate_scaled_wrap.cpp.

9.49 traits_asptr< Type > Struct Template Reference

Static Public Member Functions

- static int **asptr** (PyObject *obj, Type **val)

9.49.1 Detailed Description

template<class Type>
struct swig::traits_asptr< Type >

Definition at line 3392 of file model_steadystate_scaled_wrap.cpp.

9.50 traits_asval< Type > Struct Template Reference

Static Public Member Functions

- static int **asval** (PyObject *obj, Type *val)

9.50.1 Detailed Description

template<class Type>
struct swig::traits_asval< Type >

Definition at line 3410 of file model_steadystate_scaled_wrap.cpp.

9.51 traits_asval< PyObject *> Struct Template Reference

Public Types

- typedef PyObject * **value_type**

Static Public Member Functions

- static int **asval** (PyObject *obj, value_type *val)

9.51.1 Detailed Description

template<>
struct swig::traits_asval< PyObject *>

Definition at line 3596 of file model_steadystate_scaled_wrap.cpp.

9.52 traits_asval< Type * > Struct Template Reference

Static Public Member Functions

- static int **asval** (PyObject *obj, Type **val)

9.52.1 Detailed Description

template<class Type>
struct swig::traits_asval< Type * >

Definition at line 3433 of file model_steadystate_scaled_wrap.cpp.

9.53 traits_check< Type, Category > Struct Template Reference

9.53.1 Detailed Description

template<class Type, class Category>
struct swig::traits_check< Type, Category >

Definition at line 3344 of file model_steadystate_scaled_wrap.cpp.

9.54 traits_check< PyObject *, value_category > Struct Template Reference

Static Public Member Functions

- static bool **check** (PyObject *)

9.54.1 Detailed Description

template<>
struct swig::traits_check< PyObject *, value_category >

Definition at line 3605 of file model_steadystate_scaled_wrap.cpp.

9.55 traits_check< Type, pointer_category > Struct Template Reference

Static Public Member Functions

- static bool **check** (PyObject *obj)

9.55.1 Detailed Description

template<class Type>
struct swig::traits_check< Type, pointer_category >

Definition at line 3526 of file model_steadystate_scaled_wrap.cpp.

9.56 traits_check< Type, value_category > Struct Template Reference

Static Public Member Functions

- static bool **check** (PyObject *obj)

9.56.1 Detailed Description

template<class Type>
struct swig::traits_check< Type, value_category >

Definition at line 3518 of file model_steadystate_scaled_wrap.cpp.

9.57 traits_from< Type > Struct Template Reference

Static Public Member Functions

- static PyObject * **from** (const Type &val)

9.57.1 Detailed Description

template<class Type>
struct swig::traits_from< Type >

Definition at line 3359 of file model_steadystate_scaled_wrap.cpp.

9.58 traits_from< const Type * > Struct Template Reference

Static Public Member Functions

- static PyObject * **from** (const Type *val)

9.58.1 Detailed Description

template<class Type>
struct swig::traits_from< const Type * >

Definition at line 3371 of file model_steadystate_scaled_wrap.cpp.

9.59 traits_from< PyObject * > Struct Template Reference

Public Types

- typedef PyObject * **value_type**

Static Public Member Functions

- static PyObject * **from** (const value_type &val)

9.59.1 Detailed Description

template<>
struct swig::traits_from< PyObject * >

Definition at line 3611 of file model_steadystate_scaled_wrap.cpp.

9.60 traits_from< Type * > Struct Template Reference

Static Public Member Functions

- static PyObject * **from** (Type *val)

9.60.1 Detailed Description

template<class Type>
struct swig::traits_from< Type * >

Definition at line 3365 of file model_steadystate_scaled_wrap.cpp.

9.61 traits_from_ptr< Type > Struct Template Reference

Static Public Member Functions

- static PyObject * **from** (Type *val, int owner=0)

9.61.1 Detailed Description

template<class Type>
struct swig::traits_from_ptr< Type >

Definition at line 3353 of file model_steadystate_scaled_wrap.cpp.

9.62 traits_info< Type > Struct Template Reference

Static Public Member Functions

- static swig_type_info * **type_query** (std::string name)
- static swig_type_info * **type_info** ()

9.62.1 Detailed Description

template<class Type>
struct swig::traits_info< Type >

Definition at line 3294 of file model_steadystate_scaled_wrap.cpp.

9.63 traits_info< Type * > Struct Template Reference

Static Public Member Functions

- static swig_type_info * **type_query** (std::string name)
- static swig_type_info * **type_info** ()

9.63.1 Detailed Description

template<class Type>
struct swig::traits_info< Type * >

Definition at line 3308 of file model_steadystate_scaled_wrap.cpp.

9.64 traits_reserve< Sequence > Struct Template Reference

Static Public Member Functions

- static void **reserve** (Sequence &, typename Sequence::size_type)

9.64.1 Detailed Description

template<class Sequence>
struct swig::traits_reserve< Sequence >

Definition at line 3699 of file model_steadystate_scaled_wrap.cpp.

9.65 value_category Struct Reference

9.65.1 Detailed Description

Definition at line 3282 of file model_steadystate_scaled_wrap.cpp.

9.66 swig_cast_info Struct Reference

Public Attributes

- swig_type_info ∗ **type**
- swig_converter_func **converter**
- struct swig_cast_info ∗ **next**
- struct swig_cast_info ∗ **prev**

9.66.1 Detailed Description

Definition at line 371 of file model_steadystate_scaled_wrap.cpp.

9.67 swig_const_info Struct Reference

Public Attributes

- int **type**
- char ∗ **name**
- long **lvalue**
- double **dvalue**
- void ∗ **pvalue**
- swig_type_info ∗∗ **ptype**

9.67.1 Detailed Description

Definition at line 1137 of file model_steadystate_scaled_wrap.cpp.

9.68 swig_globalvar Struct Reference

Public Attributes

- char ∗ **name**
- PyObject ∗(∗ **get_attr**)(void)
- int(∗ **set_attr**)(PyObject ∗)
- struct swig_globalvar ∗ **next**

9.68.1 Detailed Description

Definition at line 4648 of file model_steadystate_scaled_wrap.cpp.

9.69 swig_module_info Struct Reference

Public Attributes

- swig_type_info ∗∗ **types**
- size_t **size**
- struct swig_module_info ∗ **next**
- swig_type_info ∗∗ **type_initial**
- swig_cast_info ∗∗ **cast_initial**
- void ∗ **clientdata**

9.69.1 Detailed Description

Definition at line 381 of file model_steadystate_scaled_wrap.cpp.

9.70 swig_dycast_func Struct Reference

Public Attributes

- const char ∗ **name**
- const char ∗ **str**
- swig_dycast_func **dcast**
- struct swig_cast_info ∗ **cast**
- void ∗ **clientdata**
- int **owndata**

9.70.1 Detailed Description

Definition at line 361 of file model_steadystate_scaled_wrap.cpp.

9.71 swig_varlinkobject Struct Reference

Public Attributes

- PyObject_HEAD swig_globalvar ∗ **vars**

9.71.1 Detailed Description

Definition at line 4655 of file model_steadystate_scaled_wrap.cpp.

9.72 SwigPyClientData Struct Reference

Public Attributes

- PyObject ∗ **klass**
- PyObject ∗ **newraw**
- PyObject ∗ **newargs**
- PyObject ∗ **destroy**
- int **delargs**
- int **implicitconv**
- PyTypeObject ∗ **pytype**

9.72.1 Detailed Description

Definition at line 1452 of file model_steadystate_scaled_wrap.cpp.

9.73 SwigPyObject Struct Reference

Public Attributes

- PyObject_HEAD void * **ptr**
- swig_type_info * **ty**
- int **own**
- PyObject * **next**

9.73.1 Detailed Description

Definition at line 1540 of file model_steadystate_scaled_wrap.cpp.

9.74 SwigPyPacked Struct Reference

Public Attributes

- PyObject_HEAD void * **pack**
- swig_type_info * **ty**
- size_t **size**

9.74.1 Detailed Description

Definition at line 2036 of file model_steadystate_scaled_wrap.cpp.

10 File Documentation

10.1 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:

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,...)

10.1.1 Macro Definition Documentation

10.1.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

MS definition of PI and other constants

Definition at line 23 of file amici.cpp.

10.1.1.2 M_PI

```
#define M_PI 3.14159265358979323846
```

define PI if we still have no definition

Definition at line 27 of file amici.cpp.

10.1.2 Function Documentation

10.1.2.1 runAmiciSimulation()

```
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.

10.1.2.2 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 122 of file amici.cpp.

10.1.2.3 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 143 of file amici.cpp.

10.2 cblas.cpp File Reference

BLAS routines required by AMICI.

```
#include "amici/cblas.h"
```

Include dependency graph for cblas.cpp:

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 int N, const double alpha, const double ∗A, const int lda, const double ∗X, const int incX, const double beta, double ∗Y, const int incY)
- void **amici_daxpy** (int n, double alpha, const double ∗x, const int incx, double ∗y, int incy)

  Compute y = a∗x + y.

10.2.1 Function Documentation

10.2.1.1 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

Parameters

	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.

10.2.1.2 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.

10.2.1.3 amici_daxpy()

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

Parameters

n	number of elements in y
alpha	scalar coefficient of x
x	vector of length n∗incx
incx	x stride
y	vector of length n∗incy
incy	y stride

Definition at line 90 of file cblas.cpp.

10.3 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:

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

  Compute y = a∗x + y.
- 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[ ])

10.3.1 Detailed Description

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

10.3.2 Function Documentation

10.3.2.1 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 296 of file interface_matlab.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

10.3.2.2 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.

Here is the call graph for this function:

Here is the caller graph for this function:

10.3.2.3 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.

10.3.2.4 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.

10.3.2.5 amici_daxpy()

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

Parameters

n	number of elements in y
alpha	scalar coefficient of x
x	vector of length n∗incx
incx	x stride
y	vector of length n∗incy
incy	y stride

Definition at line 90 of file cblas.cpp.

10.3.2.6 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 171 of file interface_matlab.cpp.

10.3.2.7 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 302 of file interface_matlab.cpp.

10.3.2.8 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 379 of file interface_matlab.cpp.

10.3.2.9 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 515 of file interface_matlab.cpp.

10.4 spline.cpp File Reference

definition of spline functions

Functions

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

  Evaluate the cubic spline function.
- double **sinteg** (int n, double u, double x[ ], double y[ ], double b[ ], double c[ ], double d[ ])

10.4.1 Detailed Description

Author

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

10.4.2 Function Documentation

10.4.2.1 spline()

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

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

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

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

Parameters

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

Return values

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

Notes

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

Definition at line 16 of file spline.cpp.

10.4.2.2 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.

10.4.2.3 sinteg()

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

Integrate the cubic spline function

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

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

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

Parameters

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

Returns

the value of the spline function at u

Notes

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

Definition at line 256 of file spline.cpp.

10.5 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:

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,...)

10.5.1 Detailed Description

This file contains definitions of various symbolic functions which

10.5.2 Function Documentation

10.5.2.1 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.

10.5.2.2 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.

10.5.2.3 getNaN()

```
double getNaN ( )
```

function returning nan

Returns

NaN

Definition at line 52 of file symbolic_functions.cpp.

10.5.2.4 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.

10.5.2.5 dirac()

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

c implementation of matlab function dirac

Parameters

x	argument

Returns

if(x==0) then INF else 0

Definition at line 77 of file symbolic_functions.cpp.

10.5.2.6 heaviside()

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

c implementation of matlab function heaviside

Parameters

x	argument

Returns

if(x>0) then 1 else 0

Definition at line 92 of file symbolic_functions.cpp.

10.5.2.7 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.

10.5.2.8 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.

10.5.2.9 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.

10.5.2.10 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.

10.5.2.11 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.

10.5.2.12 spline()

```
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.

10.5.2.13 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.

10.5.2.14 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.

10.5.2.15 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.

10.5.2.16 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.

10.5.2.17 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.

Index

_USE_MATH_DEFINES
amici.cpp, 404
~Model
amici::Model, 79

adjInit
amici::Solver, 320
AmiException, 15
amici::AmiException, 15, 16
AmiVector, 17
amici::AmiVector, 18, 19
AmiVectorArray, 24
amici::AmiVectorArray, 25
amici.cpp, 403
_USE_MATH_DEFINES, 404
M_PI, 404
printErrMsgIdAndTxt, 405
printWarnMsgIdAndTxt, 405
runAmiciSimulation, 404
amici::AmiException
AmiException, 15, 16
getBacktrace, 16
storeBacktrace, 17
what, 16
amici::AmiVector
AmiVector, 18, 19
at, 24
data, 19, 20
getLength, 22
getNVector, 20
getVector, 21
minus, 22
operator=, 19
operator[], 23
reset, 22
set, 23
amici::AmiVectorArray
AmiVectorArray, 25
at, 27
data, 26, 27
getLength, 29
getNVector, 28
getNVectorArray, 28
operator[], 29
reset, 30
amici::BackwardProblem
BackwardProblem, 31
getdJydx, 36
getdxBptr, 35
gett, 33
getwhich, 34
getwhichptr, 34
getxBptr, 34
getxQBptr, 35
workBackwardProblem, 32
amici::CvodeException
CvodeException, 37
amici::ExpData
ExpData, 39–42
fixedParameters, 57
fixedParametersPreequilibration, 57
getObservedData, 46
getObservedDataPtr, 47
getObservedDataStdDev, 51
getObservedDataStdDevPtr, 51
getObservedEvents, 53
getObservedEventsPtr, 53
getObservedEventsStdDev, 56
getObservedEventsStdDevPtr, 56
getTimepoint, 44
getTimepoints, 44
isSetObservedData, 46
isSetObservedDataStdDev, 50
isSetObservedEvents, 52
isSetObservedEventsStdDev, 55
nmaxevent, 57
nt, 42
nytrue, 57
nztrue, 57
observedData, 58
observedDataStdDev, 58
observedEvents, 58
observedEventsStdDev, 58
setObservedData, 45
setObservedDataStdDev, 47, 49, 50
setObservedEvents, 52
setObservedEventsStdDev, 54, 55
setTimepoints, 43
ts, 58
amici::ForwardProblem
edata, 67
ForwardProblem, 59
getDJydx, 64
getDJzdx, 64
getDiscontinuities, 63
getNumberOfRoots, 63
getRHSAtDiscontinuities, 62
getRHSBeforeDiscontinuities, 63
getRootCounter, 64
getRootIndexes, 63
getStateDerivativePointer, 65
getStateDerivativeSensitivityPointer, 66
getStatePointer, 65
getStateSensitivity, 62
getStateSensitivityPointer, 66
getStatesAtDiscontinuities, 62
getTime, 61
model, 67
rdata, 67
solver, 67

workForwardProblem, 60
amici::IDAException
IDAException, 68
amici::IntegrationFailure
error_code, 70
IntegrationFailure, 70
time, 70
amici::Model
~Model, 79
boost::serialization::serialize, 172
checkFinite, 139
clone, 79
dJrzdsigma, 180
dJrzdz, 180
dJydp, 177
dJydsigma, 179
dJydy, 179
dJzdp, 177
dJzdsigma, 180
dJzdz, 179
deltaqB, 178
deltasx, 178
deltax, 177
deltaxB, 178
drzdp, 181
drzdx, 181
dsigmaydp, 176
dsigmazdp, 177
dwdp, 182
dwdx, 182
dxdotdp, 178
dydp, 181
dydx, 181
dzdp, 180
dzdx, 180
fFIM, 137
fJDiag, 83
fJSparse, 82
fJrz, 104, 160
fJv, 84
fJy, 103, 159
fJz, 103, 159
fchi2, 136
fdJrzdsigma, 108, 163
fdJrzdz, 107, 162
fdJydp, 110
fdJydsigma, 105, 161
fdJydx, 111
fdJydy, 105, 160
fdJzdp, 112
fdJzdsigma, 107, 162
fdJzdx, 113
fdJzdz, 106, 161
fdeltaqB, 99, 156
fdeltasx, 97, 155
fdeltax, 96, 155
fdeltaxB, 98, 156
fdrzdp, 95, 154
fdrzdx, 96, 154
fdsigmaydp, 100, 158
fdsigmazdp, 102, 158
fdwdp, 133, 164
fdwdx, 134, 164
fdx0, 85
fdxdotdp, 83
fdydp, 89, 149
fdydx, 89, 150
fdzdp, 94, 152
fdzdx, 95, 153
fixedParameters, 183
fJ, 81
fres, 136
froot, 80
frz, 92, 151
fsJy, 110
fsJz, 112
fsdx0, 87
fsigmay, 100, 157
fsigmaz, 101, 158
fsres, 136
fsrz, 93, 152
fstau, 87, 148
fsx0, 86, 148
fsy, 109
fsz, 91, 151
fsz_tf, 109
fw, 131, 163
fx0, 85, 147
fxdot, 81
fy, 88, 149
fz, 90, 150
getFixedParameterIds, 146
getFixedParameterNames, 142
getFixedParameters, 123
getInitialStateSensitivities, 128
getInitialStates, 127
getObservableIds, 147
getObservableNames, 143
getParameterIds, 144
getParameterList, 126
getParameterNames, 140
getParameterScale, 120
getParameters, 122
getSolver, 80
getStateIds, 145
getStateNames, 141
getTimepoints, 124
getUnscaledParameters, 122
getmy, 165
getmz, 166
getrz, 170
getsrz, 171
getsx, 169
getsz, 171
gett, 138
getx, 168

gety, 167
getz, 169
h, 182
hasFixedParameterIds, 145
hasFixedParameterNames, 141
hasObservableIds, 146
hasObservableNames, 142
hasParameterIds, 143
hasParameterNames, 139
hasStateIds, 144
hasStateNames, 140
idlist, 176
initHeaviside, 115
initialize, 114
initializeStates, 115
J, 178
k, 118
lbw, 175
M, 182
Model, 77, 78
my, 179
mz, 179
nMaxEvent, 120
ndwdp, 175
ndwdx, 174
ne, 174
nJ, 175
nk, 118
nmaxevent, 184
nnz, 175
np, 117
nplist, 116
nt, 120
nw, 174
nx, 173
nxtrue, 173
ny, 173
nytrue, 173
nz, 174
nztrue, 174
o2mode, 176
operator=, 79
operator==, 172
originalParameters, 183
plist, 130
plist_, 183
pscale, 184
setFixedParameters, 123
setInitialStateSensitivities, 128
setInitialStates, 128
setNMaxEvent, 120
setParameterList, 127
setParameterScale, 121
setParameters, 122
setT0, 129
setTimepoints, 124
sigmay, 176
sigmaz, 177
stau, 182
sx0data, 184
t, 125
t0, 129
ts, 184
tstart, 184
ubw, 175
unscaleParameters, 130
unscaledParameters, 183
updateHeaviside, 138
updateHeavisideB, 138
w, 181
x0data, 183
z2event, 176
amici::Model_DAE
fJDiag, 193, 207
fJSparse, 190, 191, 206
fJSparseB, 192, 206
fJB, 189, 205
fJv, 193, 194, 208
fJvB, 195, 208
fdxdotdp, 201, 202, 211
fJ, 188, 189, 204
fM, 203, 213
fqBdot, 200, 211
froot, 196, 197, 209
fsxdot, 202, 212
fxBdot, 199, 210
fxdot, 198, 199, 210
getSolver, 204
Model_DAE, 186
amici::Model_ODE
fJDiag, 222, 223, 236
fJSparse, 219, 220, 235
fJSparseB, 221, 236
fJB, 219, 235
fJv, 224, 225, 237
fJvB, 226, 237
fdxdotdp, 231, 232, 240
fJ, 217, 218, 234
fqBdot, 230, 239
froot, 226, 227, 238
fsxdot, 233, 241
fxBdot, 230, 239
fxdot, 228, 229, 238
getSolver, 234
Model_ODE, 215, 216
amici::NewtonFailure
NewtonFailure, 242
amici::NewtonSolver
atol, 248
dx, 249
getSensis, 246
getSolver, 244
getStep, 245
maxlinsteps, 248
maxsteps, 248
model, 248

NewtonSolver, 244
prepareLinearSystem, 246
rdata, 249
rtol, 248
solveLinearSystem, 247
t, 248
x, 249
xdot, 249
amici::NewtonSolverDense
NewtonSolverDense, 250
prepareLinearSystem, 251
solveLinearSystem, 251
amici::NewtonSolverIterative
linsolveSPBCG, 255
NewtonSolverIterative, 253
prepareLinearSystem, 254
solveLinearSystem, 254
amici::NewtonSolverSparse
NewtonSolverSparse, 257
prepareLinearSystem, 258
solveLinearSystem, 258
amici::ReturnData
applyChainRuleFactorToSimulationResults, 263
boost::serialization::serialize, 264
chi2, 271
FIM, 268
invalidate, 261
invalidateLLH, 263
J, 265
llh, 271
ne, 274
newton_maxsteps, 275
newton_numlinsteps, 270
newton_numsteps, 270
newton_status, 270
newton_time, 270
nJ, 274
nk, 272
nmaxevent, 274
np, 272
nplist, 274
nt, 274
numerrtestfails, 269
numerrtestfailsB, 269
numnonlinsolvconvfails, 269
numnonlinsolvconvfailsB, 269
numrhsevals, 268
numrhsevalsB, 269
numsteps, 268
numstepsB, 268
nx, 272
nxtrue, 273
ny, 273
nytrue, 273
nz, 273
nztrue, 273
o2mode, 275
order, 270
pscale, 275
res, 267
ReturnData, 261
rz, 266
s2llh, 272
s2rz, 266
sensi, 275
sensi_meth, 275
sigmay, 267
sigmaz, 265
sllh, 271
sres, 268
srz, 266
ssigmay, 267
ssigmaz, 265
status, 272
sx, 266
sx0, 271
sy, 267
sz, 265
ts, 264
x, 266
x0, 271
xdot, 264
y, 267
z, 265
amici::SetupFailure
SetupFailure, 276
amici::Solver
adjInit, 320
band, 323
bandB, 324
binit, 307
boost::serialization::serialize, 338
calcICB, 290
calcIC, 288
clone, 280
create, 313
createB, 321
dense, 322
denseB, 323
diag, 324
diagB, 325
getAbsoluteTolerance, 299
getAbsoluteToleranceQuadratures, 301
getAdjBmem, 333
getDiagnosis, 285
getDiagnosisB, 286
getDky, 320
getInternalSensitivityMethod, 306
getInterpolationType, 303
getLastOrder, 332
getLinearMultistepMethod, 302
getLinearSolver, 305
getMaxSteps, 301
getMaxStepsBackwardProblem, 302
getNewtonMaxLinearSteps, 298
getNewtonMaxSteps, 296

getNewtonPreequilibration, 297
getNonlinearSolverIteration, 303
getNumErrTestFails, 331
getNumNonlinSolvConvFails, 332
getNumRhsEvals, 331
getNumSteps, 330
getQuadB, 294
getRelativeTolerance, 299
getRelativeToleranceQuadratures, 300
getRootInfo, 287
getSens, 285
getSensitivityMethod, 296
getSensitivityOrder, 298
getStabilityLimitFlag, 305
getStateOrdering, 304
getB, 293
init, 306
initializeLinearSolver, 334
initializeLinearSolverB, 336
klu, 328
kluSetOrdering, 329
kluSetOrderingB, 329
kluB, 330
operator==, 338
qbinit, 308
quadReInitB, 295
quadSStolerancesB, 322
reInit, 287
reInitB, 293
rootInit, 308
sensInit1, 309
sensReInit, 288
setAbsoluteTolerance, 300
setAbsoluteToleranceQuadratures, 301
setBandJacFn, 310
setBandJacFnB, 311
setDenseJacFn, 309
setDenseJacFnB, 310
setErrHandlerFn, 315
setId, 318
setInternalSensitivityMethod, 306
setInterpolationType, 304
setJacTimesVecFn, 310
setJacTimesVecFnB, 312
setLinearMultistepMethod, 302
setLinearSolver, 306
setMaxNumSteps, 316
setMaxNumStepsB, 317
setMaxSteps, 301
setMaxStepsBackwardProblem, 302
setNewtonMaxLinearSteps, 298
setNewtonMaxSteps, 297
setNewtonPreequilibration, 297
setNonlinearSolverIteration, 303
setQuadErrConB, 315
setRelativeTolerance, 299
setRelativeToleranceQuadratures, 300
setSStolerances, 313
setSStolerancesB, 321
setSensErrCon, 314
setSensParams, 319
setSensSStolerances, 314
setSensitivityMethod, 296
setSensitivityOrder, 298
setSparseJacFn, 310
setSparseJacFnB, 311
setStabLimDet, 317
setStabLimDetB, 318
setStabilityLimitFlag, 305
setStateOrdering, 304
setStopTime, 292
setSuppressAlg, 319
setUserData, 316
setUserDataB, 316
setup, 281
setupAMIB, 283
solve, 290
solveB, 292
solveF, 291
Solver, 280
solverMemory, 338
solverWasCalled, 339
spbcg, 326
spbcgB, 327
spgmr, 325
spgmrB, 326
sptfqmr, 327
sptfqmrB, 328
turnOffRootFinding, 295
wrapErrHandlerFn, 312
amici::SteadystateProblem
applyNewtonsMethod, 341
createSteadystateSimSolver, 345
getNewtonOutput, 342
getSteadystateSimulation, 343
SteadystateProblem, 339
workSteadyStateProblem, 340
amici_blasCBlasTransToBlasTrans
interface_matlab.cpp, 411
amici_daxpy
cblas.cpp, 409
interface_matlab.cpp, 415
amici_dgemm
cblas.cpp, 406
interface_matlab.cpp, 412
amici_dgemv
cblas.cpp, 408
interface_matlab.cpp, 413
applyChainRuleFactorToSimulationResults
amici::ReturnData, 263
applyNewtonsMethod
amici::SteadystateProblem, 341
at
amici::AmiVector, 24
amici::AmiVectorArray, 27
atol

amici::NewtonSolver, 248

BackwardProblem, 30
amici::BackwardProblem, 31
band
amici::Solver, 323
bandB
amici::Solver, 324
binit
amici::Solver, 307
boost::serialization::serialize
amici::Model, 172
amici::ReturnData, 264
amici::Solver, 338

calcICB
amici::Solver, 290
calcIC
amici::Solver, 288
cblas.cpp, 406
amici_daxpy, 409
amici_dgemm, 406
amici_dgemv, 408
checkFinite
amici::Model, 139
chi2
amici::ReturnData, 271
clone
amici::Model, 79
amici::Solver, 280
Model_model_steadystate_scaled, 349
create
amici::Solver, 313
createSteadystateSimSolver
amici::SteadystateProblem, 345
createB
amici::Solver, 321
CvodeException, 37
amici::CvodeException, 37

DDspline
symbolic_functions.cpp, 427
DDspline_pos
symbolic_functions.cpp, 428
dJrzdsigma
amici::Model, 180
dJrzdz
amici::Model, 180
dJydp
amici::Model, 177
dJydsigma
amici::Model, 179
dJydy
amici::Model, 179
dJzdp
amici::Model, 177
dJzdsigma
amici::Model, 180
dJzdz
amici::Model, 179
data
amici::AmiVector, 19, 20
amici::AmiVectorArray, 26, 27
dbl2int
interface_matlab.cpp, 411
deltaqB
amici::Model, 178
deltasx
amici::Model, 178
deltax
amici::Model, 177
deltaxB
amici::Model, 178
dense
amici::Solver, 322
denseB
amici::Solver, 323
diag
amici::Solver, 324
diagB
amici::Solver, 325
dirac
symbolic_functions.cpp, 422
Dmax
symbolic_functions.cpp, 424
Dmin
symbolic_functions.cpp, 425
drzdp
amici::Model, 181
drzdx
amici::Model, 181
dsigmaydp
amici::Model, 176
dsigmazdp
amici::Model, 177
Dspline
symbolic_functions.cpp, 426
Dspline_pos
symbolic_functions.cpp, 427
dwdp
amici::Model, 182
dwdx
amici::Model, 182
dx
amici::NewtonSolver, 249
dxdotdp
amici::Model, 178
dydp
amici::Model, 181
dydx
amici::Model, 181
dzdp
amici::Model, 180
dzdx
amici::Model, 180

edata
amici::ForwardProblem, 67

error_code
amici::IntegrationFailure, 70
ExpData, 38
amici::ExpData, 39–42
expDataFromMatlabCall
interface_matlab.cpp, 415

fFIM
amici::Model, 137
FIM
amici::ReturnData, 268
fJDiag
amici::Model, 83
amici::Model_DAE, 193, 207
amici::Model_ODE, 222, 223, 236
Model_model_steadystate_scaled, 351
fJSparse
amici::Model, 82
amici::Model_DAE, 190, 191, 206
amici::Model_ODE, 219, 220, 235
Model_model_steadystate_scaled, 352
fJSparseB
amici::Model_DAE, 192, 206
amici::Model_ODE, 221, 236
Model_model_steadystate_scaled, 353
fJB
amici::Model_DAE, 189, 205
amici::Model_ODE, 219, 235
Model_model_steadystate_scaled, 350
fJrz
amici::Model, 104, 160
Model_model_steadystate_scaled, 354
fJv
amici::Model, 84
amici::Model_DAE, 193, 194, 208
amici::Model_ODE, 224, 225, 237
Model_model_steadystate_scaled, 354
fJvB
amici::Model_DAE, 195, 208
amici::Model_ODE, 226, 237
Model_model_steadystate_scaled, 356
fJy
amici::Model, 103, 159
Model_model_steadystate_scaled, 357
fJz
amici::Model, 103, 159
Model_model_steadystate_scaled, 358
fchi2
amici::Model, 136
fdJrzdsigma
amici::Model, 108, 163
Model_model_steadystate_scaled, 358
fdJrzdz
amici::Model, 107, 162
Model_model_steadystate_scaled, 359
fdJydp
amici::Model, 110
fdJydsigma
amici::Model, 105, 161
Model_model_steadystate_scaled, 359
fdJydx
amici::Model, 111
fdJydy
amici::Model, 105, 160
Model_model_steadystate_scaled, 360
fdJzdp
amici::Model, 112
fdJzdsigma
amici::Model, 107, 162
Model_model_steadystate_scaled, 361
fdJzdx
amici::Model, 113
fdJzdz
amici::Model, 106, 161
Model_model_steadystate_scaled, 361
fdeltaqB
amici::Model, 99, 156
Model_model_steadystate_scaled, 362
fdeltasx
amici::Model, 97, 155
Model_model_steadystate_scaled, 362
fdeltax
amici::Model, 96, 155
Model_model_steadystate_scaled, 363
fdeltaxB
amici::Model, 98, 156
Model_model_steadystate_scaled, 364
fdrzdp
amici::Model, 95, 154
Model_model_steadystate_scaled, 364
fdrzdx
amici::Model, 96, 154
Model_model_steadystate_scaled, 365
fdsigmaydp
amici::Model, 100, 158
Model_model_steadystate_scaled, 365
fdsigmazdp
amici::Model, 102, 158
Model_model_steadystate_scaled, 366
fdwdp
amici::Model, 133, 164
Model_model_steadystate_scaled, 367
fdwdx
amici::Model, 134, 164
Model_model_steadystate_scaled, 367
fdx0
amici::Model, 85
fdxdotdp
amici::Model, 83
amici::Model_DAE, 201, 202, 211
amici::Model_ODE, 231, 232, 240
Model_model_steadystate_scaled, 368
fdydp
amici::Model, 89, 149
Model_model_steadystate_scaled, 370
fdydx
amici::Model, 89, 150

Model_model_steadystate_scaled, 369
fdzdp
amici::Model, 94, 152
Model_model_steadystate_scaled, 371
fdzdx
amici::Model, 95, 153
Model_model_steadystate_scaled, 371
fixedParameters
amici::ExpData, 57
amici::Model, 183
fixedParametersPreequilibration
amici::ExpData, 57
fJ
amici::Model, 81
amici::Model_DAE, 188, 189, 204
amici::Model_ODE, 217, 218, 234
Model_model_steadystate_scaled, 349
fM
amici::Model_DAE, 203, 213
ForwardProblem, 59
amici::ForwardProblem, 59
fqBdot
amici::Model_DAE, 200, 211
amici::Model_ODE, 230, 239
Model_model_steadystate_scaled, 372
fres
amici::Model, 136
from_oper< ValueType >, 385
froot
amici::Model, 80
amici::Model_DAE, 196, 197, 209
amici::Model_ODE, 226, 227, 238
Model_model_steadystate_scaled, 373
frz
amici::Model, 92, 151
Model_model_steadystate_scaled, 373
fsJy
amici::Model, 110
fsJz
amici::Model, 112
fsdx0
amici::Model, 87
fsigmay
amici::Model, 100, 157
Model_model_steadystate_scaled, 374
fsigmaz
amici::Model, 101, 158
Model_model_steadystate_scaled, 374
fsres
amici::Model, 136
fsrz
amici::Model, 93, 152
Model_model_steadystate_scaled, 375
fstau
amici::Model, 87, 148
Model_model_steadystate_scaled, 375
fsx0
amici::Model, 86, 148
Model_model_steadystate_scaled, 376
fsxdot
amici::Model_DAE, 202, 212
amici::Model_ODE, 233, 241
Model_model_steadystate_scaled, 377
fsy
amici::Model, 109
fsz
amici::Model, 91, 151
Model_model_steadystate_scaled, 377
fsz_tf
amici::Model, 109
fw
amici::Model, 131, 163
Model_model_steadystate_scaled, 378
fx0
amici::Model, 85, 147
Model_model_steadystate_scaled, 379
fxBdot
amici::Model_DAE, 199, 210
amici::Model_ODE, 230, 239
Model_model_steadystate_scaled, 379
fxdot
amici::Model, 81
amici::Model_DAE, 198, 199, 210
amici::Model_ODE, 228, 229, 238
Model_model_steadystate_scaled, 380
fy
amici::Model, 88, 149
Model_model_steadystate_scaled, 381
fz
amici::Model, 90, 150
Model_model_steadystate_scaled, 382

getAbsoluteTolerance
amici::Solver, 299
getAbsoluteToleranceQuadratures
amici::Solver, 301
getAdjBmem
amici::Solver, 333
getBacktrace
amici::AmiException, 16
getDJydx
amici::ForwardProblem, 64
getDJzdx
amici::ForwardProblem, 64
getDiagnosis
amici::Solver, 285
getDiagnosisB
amici::Solver, 286
getDiscontinuities
amici::ForwardProblem, 63
getDky
amici::Solver, 320
getFixedParameterIds
amici::Model, 146
getFixedParameterNames
amici::Model, 142
Model_model_steadystate_scaled, 383

getFixedParameters
amici::Model, 123
getInitialStateSensitivities
amici::Model, 128
getInitialStates
amici::Model, 127
getInternalSensitivityMethod
amici::Solver, 306
getInterpolationType
amici::Solver, 303
getLastOrder
amici::Solver, 332
getLength
amici::AmiVector, 22
amici::AmiVectorArray, 29
getLinearMultistepMethod
amici::Solver, 302
getLinearSolver
amici::Solver, 305
getMaxSteps
amici::Solver, 301
getMaxStepsBackwardProblem
amici::Solver, 302
getNVector
amici::AmiVector, 20
amici::AmiVectorArray, 28
getNVectorArray
amici::AmiVectorArray, 28
getNaN
symbolic_functions.cpp, 422
getNewtonMaxLinearSteps
amici::Solver, 298
getNewtonMaxSteps
amici::Solver, 296
getNewtonOutput
amici::SteadystateProblem, 342
getNewtonPreequilibration
amici::Solver, 297
getNonlinearSolverIteration
amici::Solver, 303
getNumErrTestFails
amici::Solver, 331
getNumNonlinSolvConvFails
amici::Solver, 332
getNumRhsEvals
amici::Solver, 331
getNumSteps
amici::Solver, 330
getNumberOfRoots
amici::ForwardProblem, 63
getObservableIds
amici::Model, 147
getObservableNames
amici::Model, 143
Model_model_steadystate_scaled, 383
getObservedData
amici::ExpData, 46
getObservedDataPtr
amici::ExpData, 47
getObservedDataStdDev
amici::ExpData, 51
getObservedDataStdDevPtr
amici::ExpData, 51
getObservedEvents
amici::ExpData, 53
getObservedEventsPtr
amici::ExpData, 53
getObservedEventsStdDev
amici::ExpData, 56
getObservedEventsStdDevPtr
amici::ExpData, 56
getParameterIds
amici::Model, 144
getParameterList
amici::Model, 126
getParameterNames
amici::Model, 140
Model_model_steadystate_scaled, 382
getParameterScale
amici::Model, 120
getParameters
amici::Model, 122
getQuadB
amici::Solver, 294
getRHSAtDiscontinuities
amici::ForwardProblem, 62
getRHSBeforeDiscontinuities
amici::ForwardProblem, 63
getRelativeTolerance
amici::Solver, 299
getRelativeToleranceQuadratures
amici::Solver, 300
getRootCounter
amici::ForwardProblem, 64
getRootIndexes
amici::ForwardProblem, 63
getRootInfo
amici::Solver, 287
getSens
amici::Solver, 285
getSensis
amici::NewtonSolver, 246
getSensitivityMethod
amici::Solver, 296
getSensitivityOrder
amici::Solver, 298
getSolver
amici::Model, 80
amici::Model_DAE, 204
amici::Model_ODE, 234
amici::NewtonSolver, 244
getStabilityLimitFlag
amici::Solver, 305
getStateDerivativePointer
amici::ForwardProblem, 65
getStateDerivativeSensitivityPointer

amici::ForwardProblem, 66
getStateIds
amici::Model, 145
getStateNames
amici::Model, 141
Model_model_steadystate_scaled, 383
getStateOrdering
amici::Solver, 304
getStatePointer
amici::ForwardProblem, 65
getStateSensitivity
amici::ForwardProblem, 62
getStateSensitivityPointer
amici::ForwardProblem, 66
getStatesAtDiscontinuities
amici::ForwardProblem, 62
getSteadystateSimulation
amici::SteadystateProblem, 343
getStep
amici::NewtonSolver, 245
getTime
amici::ForwardProblem, 61
getTimepoint
amici::ExpData, 44
getTimepoints
amici::ExpData, 44
amici::Model, 124
getUnscaledParameters
amici::Model, 122
getVector
amici::AmiVector, 21
getB
amici::Solver, 293
getdJydx
amici::BackwardProblem, 36
getdxBptr
amici::BackwardProblem, 35
getmy
amici::Model, 165
getmz
amici::Model, 166
getrz
amici::Model, 170
getsrz
amici::Model, 171
getsx
amici::Model, 169
getsz
amici::Model, 171
gett
amici::BackwardProblem, 33
amici::Model, 138
getwhich
amici::BackwardProblem, 34
getwhichptr
amici::BackwardProblem, 34
getx
amici::Model, 168
getxBptr
amici::BackwardProblem, 34
getxQBptr
amici::BackwardProblem, 35
gety
amici::Model, 167
getz
amici::Model, 169

h
amici::Model, 182
hasFixedParameterIds
amici::Model, 145
hasFixedParameterNames
amici::Model, 141
hasObservableIds
amici::Model, 146
hasObservableNames
amici::Model, 142
hasParameterIds
amici::Model, 143
hasParameterNames
amici::Model, 139
hasStateIds
amici::Model, 144
hasStateNames
amici::Model, 140
heaviside
symbolic_functions.cpp, 423

IDAException, 68
amici::IDAException, 68
idlist
amici::Model, 176
init
amici::Solver, 306
initHeaviside
amici::Model, 115
initialize
amici::Model, 114
initializeLinearSolver
amici::Solver, 334
initializeLinearSolverB
amici::Solver, 336
initializeStates
amici::Model, 115
IntegrationFailure, 69
amici::IntegrationFailure, 70
interface_matlab.cpp, 409
amici_blasCBlasTransToBlasTrans, 411
amici_daxpy, 415
amici_dgemm, 412
amici_dgemv, 413
dbl2int, 411
expDataFromMatlabCall, 415
mexFunction, 416
setModelData, 416
setSolverOptions, 416
invalidate

amici::ReturnData, 261
invalidateLLH
amici::ReturnData, 263
isInf
symbolic_functions.cpp, 421
isNaN
symbolic_functions.cpp, 421
isSetObservedData
amici::ExpData, 46
isSetObservedDataStdDev
amici::ExpData, 50
isSetObservedEvents
amici::ExpData, 52
isSetObservedEventsStdDev
amici::ExpData, 55

J
amici::Model, 178
amici::ReturnData, 265

k
amici::Model, 118
klu
amici::Solver, 328
kluSetOrdering
amici::Solver, 329
kluSetOrderingB
amici::Solver, 329
kluB
amici::Solver, 330

lbw
amici::Model, 175
less< PyObject * >, 384
less< swig::SwigPtr_PyObject >, 384
less< swig::SwigVar_PyObject >, 384
linsolveSPBCG
amici::NewtonSolverIterative, 255
llh
amici::ReturnData, 271
log
symbolic_functions.cpp, 422

M
amici::Model, 182
M_PI
amici.cpp, 404
max
symbolic_functions.cpp, 423
maxlinsteps
amici::NewtonSolver, 248
maxsteps
amici::NewtonSolver, 248
mexFunction
interface_matlab.cpp, 416
min
symbolic_functions.cpp, 424
minus
amici::AmiVector, 22
Model, 71
amici::Model, 77, 78
model
amici::ForwardProblem, 67
amici::NewtonSolver, 248
Model_DAE, 185
amici::Model_DAE, 186
Model_ODE, 213
amici::Model_ODE, 215, 216
Model_model_steadystate_scaled, 346
clone, 349
fJDiag, 351
fJSparse, 352
fJSparseB, 353
fJB, 350
fJrz, 354
fJv, 354
fJvB, 356
fJy, 357
fJz, 358
fdJrzdsigma, 358
fdJrzdz, 359
fdJydsigma, 359
fdJydy, 360
fdJzdsigma, 361
fdJzdz, 361
fdeltaqB, 362
fdeltasx, 362
fdeltax, 363
fdeltaxB, 364
fdrzdp, 364
fdrzdx, 365
fdsigmaydp, 365
fdsigmazdp, 366
fdwdp, 367
fdwdx, 367
fdxdotdp, 368
fdydp, 370
fdydx, 369
fdzdp, 371
fdzdx, 371
fJ, 349
fqBdot, 372
froot, 373
frz, 373
fsigmay, 374
fsigmaz, 374
fsrz, 375
fstau, 375
fsx0, 376
fsxdot, 377
fsz, 377
fw, 378
fx0, 379
fxBdot, 379
fxdot, 380
fy, 381
fz, 382

getFixedParameterNames, 383
getObservableNames, 383
getParameterNames, 382
getStateNames, 383
my
amici::Model, 179
mz
amici::Model, 179

nMaxEvent
amici::Model, 120
ndwdp
amici::Model, 175
ndwdx
amici::Model, 174
ne
amici::Model, 174
amici::ReturnData, 274
newton_maxsteps
amici::ReturnData, 275
newton_numlinsteps
amici::ReturnData, 270
newton_numsteps
amici::ReturnData, 270
newton_status
amici::ReturnData, 270
newton_time
amici::ReturnData, 270
NewtonFailure, 241
amici::NewtonFailure, 242
NewtonSolver, 242
amici::NewtonSolver, 244
NewtonSolverDense, 250
amici::NewtonSolverDense, 250
NewtonSolverIterative, 252
amici::NewtonSolverIterative, 253
NewtonSolverSparse, 256
amici::NewtonSolverSparse, 257
nJ
amici::Model, 175
amici::ReturnData, 274
nk
amici::Model, 118
amici::ReturnData, 272
nmaxevent
amici::ExpData, 57
amici::Model, 184
amici::ReturnData, 274
nnz
amici::Model, 175
noconst_traits< const Type >, 385
noconst_traits< Type >, 385
np
amici::Model, 117
amici::ReturnData, 272
nplist
amici::Model, 116
amici::ReturnData, 274
nt
amici::ExpData, 42
amici::Model, 120
amici::ReturnData, 274
numerrtestfails
amici::ReturnData, 269
numerrtestfailsB
amici::ReturnData, 269
numnonlinsolvconvfails
amici::ReturnData, 269
numnonlinsolvconvfailsB
amici::ReturnData, 269
numrhsevals
amici::ReturnData, 268
numrhsevalsB
amici::ReturnData, 269
numsteps
amici::ReturnData, 268
numstepsB
amici::ReturnData, 268
nw
amici::Model, 174
nx
amici::Model, 173
amici::ReturnData, 272
nxtrue
amici::Model, 173
amici::ReturnData, 273
ny
amici::Model, 173
amici::ReturnData, 273
nytrue
amici::ExpData, 57
amici::Model, 173
amici::ReturnData, 273
nz
amici::Model, 174
amici::ReturnData, 273
nztrue
amici::ExpData, 57
amici::Model, 174
amici::ReturnData, 273

o2mode
amici::Model, 176
amici::ReturnData, 275
observedData
amici::ExpData, 58
observedDataStdDev
amici::ExpData, 58
observedEvents
amici::ExpData, 58
observedEventsStdDev
amici::ExpData, 58
operator=
amici::AmiVector, 19
amici::Model, 79
operator==
amici::Model, 172
amici::Solver, 338

operator[]
amici::AmiVector, 23
amici::AmiVectorArray, 29
order
amici::ReturnData, 270
originalParameters
amici::Model, 183

plist
amici::Model, 130
plist_
amici::Model, 183
pointer_category, 386
prepareLinearSystem
amici::NewtonSolver, 246
amici::NewtonSolverDense, 251
amici::NewtonSolverIterative, 254
amici::NewtonSolverSparse, 258
printErrMsgIdAndTxt
amici.cpp, 405
printWarnMsgIdAndTxt
amici.cpp, 405
pscale
amici::Model, 184
amici::ReturnData, 275
PyHeapTypeObject, 384

qbinit
amici::Solver, 308
quadReInitB
amici::Solver, 295
quadSStolerancesB
amici::Solver, 322

rdata
amici::ForwardProblem, 67
amici::NewtonSolver, 249
reInit
amici::Solver, 287
reInitB
amici::Solver, 293
res
amici::ReturnData, 267
reset
amici::AmiVector, 22
amici::AmiVectorArray, 30
ReturnData, 259
amici::ReturnData, 261
rootInit
amici::Solver, 308
rtol
amici::NewtonSolver, 248
runAmiciSimulation
amici.cpp, 404
rz
amici::ReturnData, 266

s2llh
amici::ReturnData, 272
s2rz
amici::ReturnData, 266
sensInit1
amici::Solver, 309
sensReInit
amici::Solver, 288
sensi
amici::ReturnData, 275
sensi_meth
amici::ReturnData, 275
set
amici::AmiVector, 23
setAbsoluteTolerance
amici::Solver, 300
setAbsoluteToleranceQuadratures
amici::Solver, 301
setBandJacFn
amici::Solver, 310
setBandJacFnB
amici::Solver, 311
setDenseJacFn
amici::Solver, 309
setDenseJacFnB
amici::Solver, 310
setErrHandlerFn
amici::Solver, 315
setFixedParameters
amici::Model, 123
setId
amici::Solver, 318
setInitialStateSensitivities
amici::Model, 128
setInitialStates
amici::Model, 128
setInternalSensitivityMethod
amici::Solver, 306
setInterpolationType
amici::Solver, 304
setJacTimesVecFn
amici::Solver, 310
setJacTimesVecFnB
amici::Solver, 312
setLinearMultistepMethod
amici::Solver, 302
setLinearSolver
amici::Solver, 306
setMaxNumSteps
amici::Solver, 316
setMaxNumStepsB
amici::Solver, 317
setMaxSteps
amici::Solver, 301
setMaxStepsBackwardProblem
amici::Solver, 302
setModelData
interface_matlab.cpp, 416
setNMaxEvent
amici::Model, 120

setNewtonMaxLinearSteps
amici::Solver, 298
setNewtonMaxSteps
amici::Solver, 297
setNewtonPreequilibration
amici::Solver, 297
setNonlinearSolverIteration
amici::Solver, 303
setObservedData
amici::ExpData, 45
setObservedDataStdDev
amici::ExpData, 47, 49, 50
setObservedEvents
amici::ExpData, 52
setObservedEventsStdDev
amici::ExpData, 54, 55
setParameterList
amici::Model, 127
setParameterScale
amici::Model, 121
setParameters
amici::Model, 122
setQuadErrConB
amici::Solver, 315
setRelativeTolerance
amici::Solver, 299
setRelativeToleranceQuadratures
amici::Solver, 300
setSStolerances
amici::Solver, 313
setSStolerancesB
amici::Solver, 321
setSensErrCon
amici::Solver, 314
setSensParams
amici::Solver, 319
setSensSStolerances
amici::Solver, 314
setSensitivityMethod
amici::Solver, 296
setSensitivityOrder
amici::Solver, 298
setSolverOptions
interface_matlab.cpp, 416
setSparseJacFn
amici::Solver, 310
setSparseJacFnB
amici::Solver, 311
setStabLimDet
amici::Solver, 317
setStabLimDetB
amici::Solver, 318
setStabilityLimitFlag
amici::Solver, 305
setStateOrdering
amici::Solver, 304
setStopTime
amici::Solver, 292
setSuppressAlg
amici::Solver, 319
setT0
amici::Model, 129
setTimepoints
amici::ExpData, 43
amici::Model, 124
setUserData
amici::Solver, 316
setUserDataB
amici::Solver, 316
setup
amici::Solver, 281
setupAMIB
amici::Solver, 283
SetupFailure, 276
amici::SetupFailure, 276
seval
spline.cpp, 419
sigmay
amici::Model, 176
amici::ReturnData, 267
sigmaz
amici::Model, 177
amici::ReturnData, 265
sign
symbolic_functions.cpp, 423
sinteg
spline.cpp, 419
sllh
amici::ReturnData, 271
solve
amici::Solver, 290
solveLinearSystem
amici::NewtonSolver, 247
amici::NewtonSolverDense, 251
amici::NewtonSolverIterative, 254
amici::NewtonSolverSparse, 258
solveB
amici::Solver, 292
solveF
amici::Solver, 291
Solver, 277
amici::Solver, 280
solver
amici::ForwardProblem, 67
solverMemory
amici::Solver, 338
solverWasCalled
amici::Solver, 339
spbcg
amici::Solver, 326
spbcgB
amici::Solver, 327
spgmr
amici::Solver, 325
spgmrB
amici::Solver, 326

spline
spline.cpp, 417
symbolic_functions.cpp, 425
spline.cpp, 417
seval, 419
sinteg, 419
spline, 417
spline_pos
symbolic_functions.cpp, 426
sptfqmr
amici::Solver, 327
sptfqmrB
amici::Solver, 328
sres
amici::ReturnData, 268
srz
amici::ReturnData, 266
ssigmay
amici::ReturnData, 267
ssigmaz
amici::ReturnData, 265
status
amici::ReturnData, 272
stau
amici::Model, 182
SteadystateProblem, 339
amici::SteadystateProblem, 339
stop_iteration, 386
storeBacktrace
amici::AmiException, 17
swig_cast_info, 401
swig_const_info, 401
swig_dycast_func, 402
swig_globalvar, 401
swig_module_info, 402
swig_varlinkobject, 402
SwigPtr_PyObject, 386
SwigPyClientData, 402
SwigPyIterator, 387
SwigPyIterator_T< OutIterator >, 388
SwigPyIteratorClosed_T< OutIterator, ValueType, FromOper >, 389
SwigPyIteratorOpen_T< OutIterator, ValueType, FromOper >, 390
SwigPyObject, 403
SwigPyPacked, 403
SwigPySequence_ArrowProxy< T >, 391
SwigPySequence_Cont< T >, 391
SwigPySequence_InputIterator< T, Reference >, 392
SwigPySequence_Ref< T >, 393
SwigVar_PyObject, 393
sx
amici::ReturnData, 266
sx0
amici::ReturnData, 271
sx0data
amici::Model, 184
sy
amici::ReturnData, 267
symbolic_functions.cpp, 420
DDspline, 427
DDspline_pos, 428
dirac, 422
Dmax, 424
Dmin, 425
Dspline, 426
Dspline_pos, 427
getNaN, 422
heaviside, 423
isInf, 421
isNaN, 421
log, 422
max, 423
min, 424
sign, 423
spline, 425
spline_pos, 426
sz
amici::ReturnData, 265

t
amici::Model, 125
amici::NewtonSolver, 248
t0
amici::Model, 129
time
amici::IntegrationFailure, 70
traits< PyObject * >, 394
traits< Type >, 394
traits< Type * >, 394
traits_as< Type *, pointer_category >, 395
traits_as< Type, Category >, 395
traits_as< Type, pointer_category >, 395
traits_as< Type, value_category >, 396
traits_asptr< Type >, 396
traits_asval< PyObject *>, 396
traits_asval< Type >, 396
traits_asval< Type * >, 397
traits_check< PyObject *, value_category >, 397
traits_check< Type, Category >, 397
traits_check< Type, pointer_category >, 398
traits_check< Type, value_category >, 398
traits_from< const Type * >, 398
traits_from< PyObject * >, 399
traits_from< Type >, 398
traits_from< Type * >, 399
traits_from_ptr< Type >, 399
traits_info< Type >, 400
traits_info< Type * >, 400
traits_reserve< Sequence >, 400
ts
amici::ExpData, 58
amici::Model, 184
amici::ReturnData, 264
tstart
amici::Model, 184
turnOffRootFinding

amici::Solver, 295

ubw
amici::Model, 175
unscaleParameters
amici::Model, 130
unscaledParameters
amici::Model, 183
updateHeaviside
amici::Model, 138
updateHeavisideB
amici::Model, 138

value_category, 401

w
amici::Model, 181
what
amici::AmiException, 16
workBackwardProblem
amici::BackwardProblem, 32
workForwardProblem
amici::ForwardProblem, 60
workSteadyStateProblem
amici::SteadystateProblem, 340
wrapErrHandlerFn
amici::Solver, 312

x
amici::NewtonSolver, 249
amici::ReturnData, 266
x0
amici::ReturnData, 271
x0data
amici::Model, 183
xdot
amici::NewtonSolver, 249
amici::ReturnData, 264

y
amici::ReturnData, 267

z
amici::ReturnData, 265
z2event
amici::Model, 176